Master thesis
Computing Science

Radboud University

Exploring DNS queries for privacy and security sensitive information

Author:
Huy Nguyen
s4791916

First supervisor/assessor:
Dr.ir. Harald Vranken

Internship supervisor:
Dr.ir. Joeri de Ruiter

Second assessor:
Dr.ir. Ileana Buhan

April 3, 2023

Abstract

DNS plays a vital role in our internet structure and serves as the phonebook of the internet. Every time we visit a website, DNS queries are generated and sent to various DNS servers. The majority of the time these DNS queries travel completely unencrypted. In this thesis, we research whether the contents of DNS queries can contain privacy or security sensitive information that can be traced back to a user or organisation. We do this by analyzing DNS datasets from DNS-OARC and SURF with various data analysis techniques. Results show that we can find username-password combinations, locational information such as coordinates and software information that can be traced back to an organisation. We conclude from this that DNS queries can leak unwanted sensitive information. Given these results, DNS datasets should not be publicly shared or shared with certain measures such as removing the source IP from all DNS queries.

Contents

Chapter 1

Introduction

Watching a video on YouTube. Sending an email. Browsing a news site. Playing some online video games. For most people, this is just a normal day on the internet. But how do these activities start from the perspective of a computer? Say that we are browsing the internet and want to visit a news site. The first step for a computer is to find the address of the news site that we are trying to visit. This is comparable to real life. If you are going to a restaurant, you first need to know the address of that restaurant. Likewise, a computer also requires an address before it can access a website on the internet. This address is known as an IP address and a request for an IP address is a Domain Name System (DNS) query. This query is a computer asking "where is this domain on the internet?". As billions of people have internet access this also means that billions of DNS queries are sent out every single day. The DNS protocol [1, 2] originates in 1983 when Paul Mockapetris first proposed it. Four years went by and the improved RFCs [3, 4] introduced in 1987 still form the basis of DNS as it is used today. Unfortunately, the originally proposed DNS structures did not have privacy or security in mind. It was less of a concern back then. Even to this day, the majority of DNS traffic is public [5, 6]. Despite DNS traffic being public, researchers such as Spring and Huth [7] argued that there are no privacy issues for end users as DNS queries cannot be linked to an end user when DNS data. However, Spring and Huth did not focus on the contents of a DNS query. Shortly summarised, there are two parts to a DNS query. There is the "who is sending the DNS query" part and the "what information is being requested" part. Spring and Huth focused on the former and this thesis aims to also look at the latter. We research the contents of DNS queries for privacy and security impacting information as there has been less research regarding this aspect. This thesis was done at SURF which is a cooperative association of Dutch educational and research institutions. SURF processes a great number of DNS data and wants a

better understanding of the sensitivity of this data for research purposes.

1.1 Research questions

We have the following main research question:

Main What are the privacy and security concerns of DNS queries that are exchanged between the user and DNS servers?

To help us answer this question we will first look to understand DNS privacy and security in the context of our thesis. This will help us in the next step where we analyse DNS datasets. The goal of our analysis is to look for sensitive information inside DNS queries. We have the following sub research questions.

Sub 1 What type of information in DNS queries can be considered privacy or security sensitive?

The first question helps us to understand the privacy and security problems in DNS. We seek to understand which types of information in DNS queries are potentially sensitive as DNS queries can contain various types of information. We attempt to define and narrow our scope for the context of this thesis.

Sub 2 How can data analysis techniques help us find privacy and security sensitive information in DNS queries?

After obtaining a better understanding of privacy and security concerns in DNS queries, we proceed with analysing DNS datasets. As DNS datasets can contain up to billions of DNS queries, we need efficient data analysis techniques to process these datasets.

1.2 Structure

The structure of this thesis is as follows. Chapter 2 goes over the technical background. We introduce the basics of DNS along with technical information regarding the used data analysis techniques. Chapter 3 discusses related work. Chapter 4 aims to answer our first sub research question. Chapter 5 goes over the data analysis techniques used in this thesis to answer the second sub research question. We explain which datasets we used and the data analysis techniques performed. Chapter 6 follows up with the results that were found along with some performance numbers. Chapter 7 contains the discussion and future work. Finally, we have chapter 8 with the conclusion.

Chapter 2

Technical background

We first discuss the technical preliminaries that are needed to understand this thesis. We explain the basics of DNS and some of the used data analysis techniques.

2.1 Basics of DNS

DNS is often described as the phonebook of the internet. Back in the day when physical phonebooks were more popular, you would use them to look up the phone number of a company or a person. The equivalent of this in DNS would be searching for an IP address that matches a certain domain. This procedure is also known as a DNS query request. Say that we are trying to look for "ru.nl".

1. To start, the web browser generates a DNS query for **ru.nl** and this is sent to a recursive resolver. This resolver handles the following queries that are needed to find the IP address belonging to **ru.nl**. It acts as a middleman between the client and DNS servers.
2. The recursive resolver sends the DNS query for **ru.nl** to a DNS root server.
3. The root server does not have a direct answer and will redirect the recursive resolver to a Top Level Domain (TLD) nameserver. In this case, the root server responds with a list of all **.nl** nameservers to choose from.
4. The recursive resolver chooses a nameserver and forwards the query to the chosen **.nl** nameserver.
5. The **.nl** nameserver will redirect the recursive resolver to an authoritative server.

6. The recursive resolver sends the DNS query for **ru.nl** to an authoritative server.
7. Finally, an authoritative DNS server will answer the recursive resolver with the IP address of **ru.nl**.
8. The recursive resolver then proceeds to pass this information to the client that requested it.

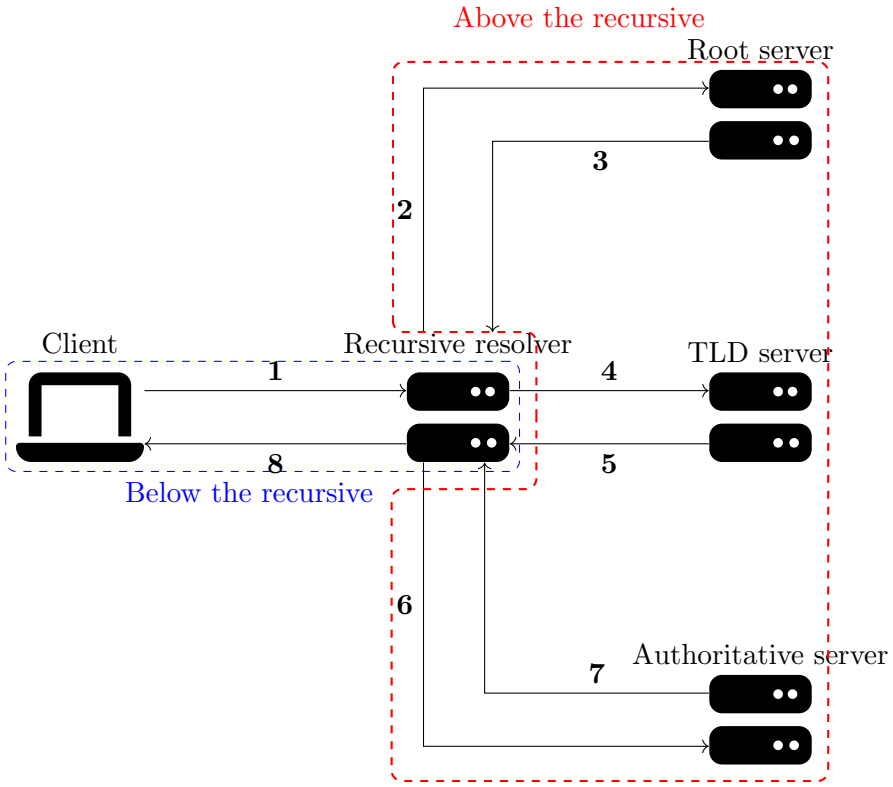


Figure 2.1: Example DNS query

It is important to realise that root, TLD and authoritative servers do not know the IP address of the client that sent the original query. The recursive resolver acts as a middleman for the client and other DNS servers. DNS queries that the recursive resolver sends on behalf of the client will have the IP address of the recursive resolver. Thus root, TLD and authoritative servers only see the IP address of the recursive resolver and not the IP address of the original client. To highlight this difference there is a blue dashed box around the client and the recursive resolver in figure 2.1. This is also known as "below the recursive". DNS queries sent between the recursive resolver and other DNS servers are referred to as "above the recursive" highlighted in red.

The example in figure 2.1 assumes that there is no caching involved. In reality, sometimes DNS queries do not have to travel through various DNS servers to find an answer. Caching of DNS records at certain places speeds up the DNS protocol. For example, let us look back at our "ru.nl" query in figure 2.1. If the recursive resolver in step 2 has "ru.nl" in its cache it will immediately reply with the IP address of "ru.nl". Caching saves time and unnecessary DNS queries. There are also other caching places such as the web browser cache.

Common day-to-day internet activities such as browsing the internet, watching a video or emailing generate DNS queries in various ways. The previously mentioned example where we searched for "ru.nl' was directly triggered by a user typing it in their browser. This is not always the case as there are instances of DNS queries generated indirectly by a user. Say that a user is visiting "ru.nl". In this case, the user actively triggers a DNS query for "ru.nl". But "ru.nl" contains images and videos hosted on other sites. A web browser must also get the IP addresses of these hosting sites to be able to show these videos and images. This results in indirect DNS queries generated for these hosting sites besides the "ru.nl" query typed by the user. Other examples of DNS queries that are not the result of direct user intervention are smart IoT devices that act on their own. Think of media devices or streaming services such as Netflix. Another example would be smart fridges with internet capabilities that generate DNS queries for various reasons.

We talked about how caching speeds up the DNS protocol. Sometimes, DNS queries do not need to travel to a root, TLD or authoritative server to find an answer. Caching only happens with valid DNS queries. So if a user makes a typo this will generate an invalid DNS query that evades the cache. This results in a DNS query that reaches a root, TLD or authoritative server. Another example is a wrongly configured device generating DNS queries. Imagine that we install a new printer, but we forget to remove the drivers and software of our old printer. These drivers might try to reach the old printer every time printing is performed. These kinds of queries will accidentally end up on DNS servers. However, some devices intentionally generate invalid DNS queries. An example of this is Chromium which is Google's open-source browser project. Chromium browsers generate random erroneous DNS queries as part of a security protocol to prevent DNS hijacking. To the point that APNIC noticed that almost the majority of the root DNS traffic were these DNS queries from Chromium web browsers [8].

2.2 Data analysis techniques

In this section, we go over some preliminaries for the data analysis techniques that we will be using in this thesis. The goal is to find efficient data analysis

techniques that can help us find privacy or security sensitive information in DNS datasets.

2.2.1 Word embedding and word2vec

Word embedding is a technique used to transform words and sentences into a numerical value such as a vector. An example of this is the word2vec [9] algorithm. This algorithm aims to find similar words in a dataset and was created by Google employees. The intuition behind word2vec is that words used in a similar context often have a similar semantic meaning. Figure 2.2 shows an example of this. In this example, we see five sentences that are grammatically built up similarly. We have a word followed by a verb followed by a noun. Word2vec does not need to understand the words, grammar or syntax. It is purely focused on trying to distinguish and find words that are used in a similar context. In this case, word2vec would consider the names "John/Anna/Martin" to be similar as well as the verbs "likes/hates" and nouns "cookie/cakes".

Figure 2.2: Example for word2vec

2.2.2 Clustering

Clustering is a machine learning technique that groups data into similar categories based on an evaluation. For example, clustering newspaper articles into similar categories, clustering customers into certain types for marketing and music services clustering their listeners to improve music recommendations. There are various clustering algorithms for different usages with pros and cons for every algorithm. Figure 2.3 from the scikit-learn [10] Python library highlights some clustering algorithms along with various use cases.

Method name	Parameters	Scalability	Usecase	Geometry (metric used)
K-Means	number of clusters	Very large `n_samples`, medium `n_clusters` with MiniBatch code	General-purpose, even cluster size, flat geometry, not too many clusters, inductive	Distances between points
Affinity propagation	damping, sample preference	Not scalable with n_samples	Many clusters, uneven cluster size, non-flat geometry, inductive	Graph distance (e.g. nearest-neighbor graph)
Mean-shift	bandwidth	Not scalable with `n_samples`	Many clusters, uneven cluster size, non-flat geometry, inductive	Distances between points
Spectral clustering	number of clusters	Medium `n_samples`, small `n_clusters`	Few clusters, even cluster size, non-flat geometry, transductive	Graph distance (e.g. nearest-neighbor graph)
Ward hierarchical clustering	number of clusters or distance threshold	Large `n_samples` and `n_clusters`	Many clusters, possibly connectivity constraints, transductive	Distances between points
Agglomerative clustering	number of clusters or distance threshold, linkage type, distance	Large `n_samples` and `n_clusters`	Many clusters, possibly connectivity constraints, non Euclidean distances, transductive	Any pairwise distance
DBSCAN	neighborhood size	Very large `n_samples`, medium `n_clusters`	Non-flat geometry, uneven cluster sizes, outlier removal, transductive	Distances between nearest points
OPTICS	minimum cluster membership	Very large `n_samples`, large `n_clusters`	Non-flat geometry, uneven cluster sizes, variable cluster density, outlier removal, transductive	Distances between points
Gaussian mixtures	many	Not scalable	Flat geometry, good for density estimation, inductive	Mahalanobis distances to centers
BIRCH	branching factor, threshold, optional global clusterer.	Large `n_clusters` and `n_samples`	Large dataset, outlier removal, data reduction, inductive	Euclidean distance between points
Bisecting K-Means	number of clusters	Very large `n_samples`, medium `n_clusters`	General-purpose, even cluster size, flat geometry, no empty clusters, inductive, hierarchical	Distances between points

Figure 2.3: Clustering algorithms offered by sklearn

For this thesis, we use mini-batch K-means which is a variant of K-Means. We use this variant as it is more efficient when processing bigger datasets. The K-Means clustering algorithm takes two inputs.

- The number of clusters we want to have.
- The data points that we want to divide into clusters.

K-means then intializes centroids in the data. A centroid is a center of a cluster and the number of centroids corresponds to the number of clusters defined by the user. In the beginning, it is still unknown where the center of a cluster is so a random centroid is chosen at first. K-means then proceeds to assign data points close to these centroids to a cluster. After that, it continues this process for a number of iterations to keep improving upon the results. Finally, we end up with the number of predetermined clusters and data points that belong to these clusters.

Chapter 3

Related work

Research regarding the privacy and security aspects of DNS has been mostly focused on the DNS traffic between the client and recursive resolvers. There has been less research on DNS traffic between recursive resolvers and root servers, top level domain nameservers and authoritative nameservers. This is also referred to as DNS queries above the recursive. As a reminder, figure 2.1 highlights these differences.

A recursive resolver acts as a middleman in the DNS process. Clients send a DNS query to the recursive resolver and the resolver handles all the necessary follow-up DNS requests. As a result of this, queries originating from a recursive resolver do not contain the original IP address of the client. Thus one might think there are no privacy issues here as we are not able to trace DNS queries back to the original source. Spring and Huth [7] have argued that the privacy of an end user should not be impacted if DNS traffic were to be collected travelling above the recursive. They argue that with caching, enough users behind a recursive resolver and without knowing the original source IP it is nearly impossible to find the original sender of a DNS query. While Spring and Huth focus on end-user privacy, Imana et al. [11] also look at institutional privacy. Instead of trying to find an individual user behind a DNS query, institutional privacy is higher level and tries to find the institution behind a query. For example, say that we have two companies that collude together to fix prices in their market. They both deny these accusations. However, when we analyse their DNS queries we still notice communication patterns between these two companies. This is an example of institutional privacy as we are not interested in an individual, but in the organisations behind these queries. Aside from finding out communication patterns, Imana et al. [11] also note that it might be interesting to find out whether sensitive sites were visited within a certain institution. For example, DNS queries for certain religious sites might give away the demographics

of the users behind these queries. They used the IAB tech lab content taxonomy [12] to categorize the sensitive DNS queries they found. Table 3.1 from Imana et al. highlights the type of DNS queries found in these datasets.

Category (IAB ID)	**Example**	**Ramification**
Illegal (IAB26-1)	123movies.best	demographics
Religion (IAB23)	jw.org	demographics
Dating (IAB14-1)	deaf.dating	demographics
LGBTQ+ (IAB14-3)	Igbt.foundation	demographics
Adult (IAB25-3)	pornhub.com	embarrassing
Gambling (IAB9-WS1)	topcasino.ml	embarrassing
Addiction (IAB7-42)	frn.rehab	embarrassing

Table 3.1: List of IAB content categories studied by Imana et al.

Similarly, Imana et al. [13] have also looked at other examples that might impact privacy. For example, trackable names. If someone uses a domain in a format similar to `firstname-lastname.com`, it can indicate when a certain person is active. Another brought-up example was that there might be IP addresses in DNS queries that can be privacy sensitive.

Aside from the previously mentioned privacy issues, security issues with DNS queries is also a researched topic. DNS queries can contain information that might leak used hardware or software. Hardaker [14] looked at DNS queries sent from two residential homes. He concluded that various queries could give away sensitive security information. For example, the DNS queries gave away which Linux distribution was running, the presence of Samsung hardware, virtual camera software and telemetry services being enabled on a computer. However, Hardaker also found queries where it was harder to explain their origins. For example, there were certain Microsoft-based queries without any Microsoft-based systems in the networks. Hardaker's research does show that DNS queries might unknowingly leak useful information. Think of a hacker who is trying to find vulnerabilities in a certain network. Knowing what kind of hardware or software is being used already narrows down the options.

There is also research using word2vec to detect malicious DNS queries. Satoh et al. [15] use a combination of word2vec and a clustering algorithm to identify malicious DNS queries. While the basic word2vec algorithm uses distances between words to uncover similar words, Satoh et al. mod-

ify word2vec to consider the time interval between queries instead of the distances between word vectors. The idea behind this is that similar DNS queries that occur in a certain time interval before or after a suspicious DNS query can indicate some form of malware communication. In the end, the authors were able to detect 388 malicious queries that were clustered into 3 clusters which was a success according to the researchers.

Aside from detecting malicious DNS queries, there are also researchers such as Lopez et al. [16] who focus on finding semantically similar domain names with word2vec. Think of domains that share a similar topic such as soccer blogs, government sites or public transport sites. The authors collected DNS data from a large ISP, preprocessed the data and gave this as input to word2vec. Following this, the results were manually inspected for similarity and the internet popularity ranking service Alexa [17] was also used to find similarities between certain domains. The authors found the results to be satisfactory. Two other advantages were mentioned by the authors when using the word2vec algorithm. Word2vec was scalable and did not need much manual intervention to get working.

Chapter 4

DNS privacy and security

In this chapter, we look to answer our first sub research question.

Sub 1 What type of information in DNS queries can be considered privacy or security sensitive?

We first explain privacy and security in the context of DNS. Afterwards, we go over some of the current existing security and privacy measures in DNS.

4.1 Privacy and security in the context of DNS

First of all, let us look at what kind of data there is in a DNS query. In general, DNS queries contain at least an **IP source**, **IP destination**, **QNAME** (query name) and **query type**. From a privacy and security perspective, we consider the **IP source** and **QNAME** to be the most important [5]. The **IP source** tells us where the query originates from and the **QNAME** is the information that is being requested in a DNS query. For example, this can be a domain such as the "ru.nl" example in figure 2.1 we had before.

When it comes to privacy, we first look at a definition from the GDPR [18] regarding personal data:

> **'personal data' means any information relating to an identified or identifiable natural person ('data subject'); an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person;**

How should we understand this in the context of DNS queries? Who is our "data subject" in this case? RFC 8890 [19] provides us with a definition of an end user in DNS. An end user is here defined as a human user of the internet. This could be a person who was browsing the internet, playing video games or sending emails. So any information in DNS queries that can lead back to an end user is considered to be privacy sensitive. We consider the source IP address and the QNAME in a DNS query to be the most important pieces of data from a privacy perspective. Having the source IP address means that we can potentially trace back who or which client made the DNS query. However, note that DNS queries sent from a recursive resolver to other DNS servers do not have the original source IP address. Recursive resolvers take the DNS query from a client and then replace the original source IP address with the IP address of the recursive resolver. Aside from the source IP address, the QNAME can also reveal sensitive information. For example, a query for "imac-of-huynguyen" reveals more information than a query for "google.com". The former contains a personal name while the latter is a query for a search engine.

However, personal privacy is only one aspect of privacy. The GDPR does not look beyond this aspect. It explicitly states [20] that the data protection rules do not apply to data about a company. For DNS queries, we can also consider institutional privacy as defined by Imana et al. [11]:

> **We define institutional privacy as confidentiality of digital footprints of an institution's operations and activities of its personnel as a whole.**

While the GDPR focuses on natural persons, Imana et al. look at privacy from a different perspective. They focus on the confidentiality of institutions and confidentiality here is the ability to keep information such as DNS queries undisclosed to a third party. Thus, uncovering communication patterns between two institutions would count as a privacy violation in this definition from Imana et al.

4.2 Current existing privacy and security measures

Moving on to security, we look at some security measures in DNS. The first one is **DNS encryption.** The majority of DNS queries are sent in plain text over the internet. This means that the contents of a DNS query can potentially be read by a third party. Two DNS encryption solutions for this are DNS over TLS [21] and DNS over HTTPS [22]. However, these encryption solutions only protect traffic from the client to a recursive resolver. We refer to this as DNS data below the recursive as seen in figure 2.1. Current DNS encryption solutions do not encrypt DNS traffic above the recursive. This is DNS traffic from recursive resolvers to root, TLD and authoritative

DNS servers. The reasons for this are decreased performance, a risk for new types of DDOS attacks, problems with detecting whether a DNS server supports encryption and uncertainty regarding the actual benefits [23, 24].

The next security measure we look at is **DNSSEC.** DNSSEC stands for Domain Name System Security Extensions and provides two services to DNS. Origin authentication and integrity protection [25]. Origin authentication ensures that the DNS responses we receive are from a trusted source. Integrity protection ensures that the data has not been tampered with. DNSSEC provides security against DNS attacks such as DNS spoofing [26] where an attacker introduces incorrect DNS information to redirect users to malicious websites. However, DNSSEC does not provide confidentiality. This means that the content of a DNS query is not protected from public snooping.

Finally, we take a look at **QNAME minimisation** [27]. QNAME minimisation focuses on reducing the amount of information a QNAME gives away in a DNS query. For example, we have a DNS query for "ru.nl". Sending this full QNAME "ru.nl" to the root server is not needed, as the root server only knows the answer for ".nl". This reduces the amount of information that is sent and reduces the number of servers that receive the full QNAME. Thus hopefully also reduces the risks of sending sensitive information to multiple servers. However, the full QNAME will still be sent in the end to the DNS server that is authoritative for "'ru.nl".

Summarised, we see that these privacy and security measures do not prevent DNS queries above the recursive from being snooped upon. DNS encryption is only used for DNS data below the recursive, DNSSEC does not provide confidentiality and QNAME minimisation reduces the amount of data being sent out, but eventually the full QNAME will still be out there. Due to all of this, the focus of our thesis will be on DNS data that is sent above the recursive. Another reason to look at DNS data above the recursive is that researchers such as Spring and Huth [7] considered DNS data above the recursive to have no privacy impacts, but other researchers such as Imana et al. [11, 13] pointed out some potential privacy issues nonetheless. These reasons make it interesting to look at DNS traffic above the recursive to obtain a better understanding of how privacy and security sensitive this data is.

Chapter 5

Data analysis techniques

In this chapter, we describe the methodology used to answer sub research question two.

Sub 2 How can data analysis techniques help us find privacy and security sensitive information in DNS queries?

We first describe the used DNS datasets and afterwards the data analysis techniques that we used. As DNS datasets can contain billions of DNS queries, our goal is to find data analysis techniques that are efficient and effective in detecting sensitive information within DNS queries. Figure 5.1 provides a short overview of the data analysis techniques that we discuss in this chapter.

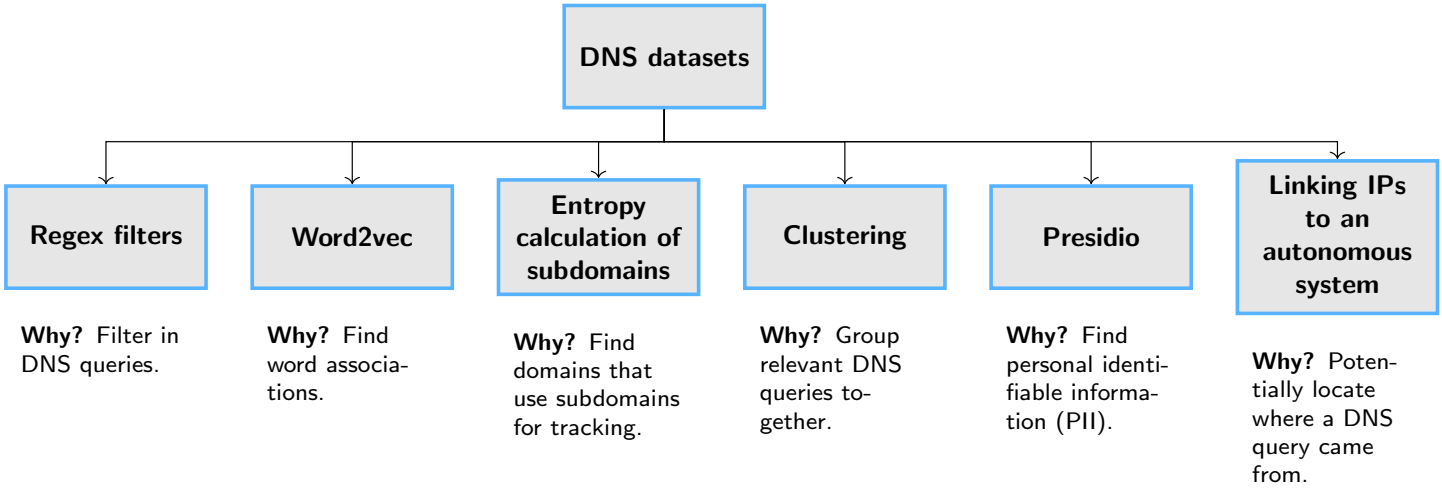


Figure 5.1: High level overview of applied data analysis techniques

5.1 Datasets

Our first step is to find DNS datasets to analyse. We turn to DNS-OARC which is a DNS research center that provides DNS data. In particular, we make use of the "Day In The Life of the Internet (DITL)" collection effort. This is DNS data that is collected from various DNS nameservers around the world every year [28]. These datasets contain data that travels from recursive resolvers to the DNS root servers. There are 13 root name servers [29] that are operated by 12 organisations. These 13 root name servers are named alphabetically from A to M. Figure 5.2 contains a world map summarizing the various root servers around the world. The numbers represent the number of instances of root servers in that region. The letters L and K are singular instances of root servers L and K and have no special meaning.

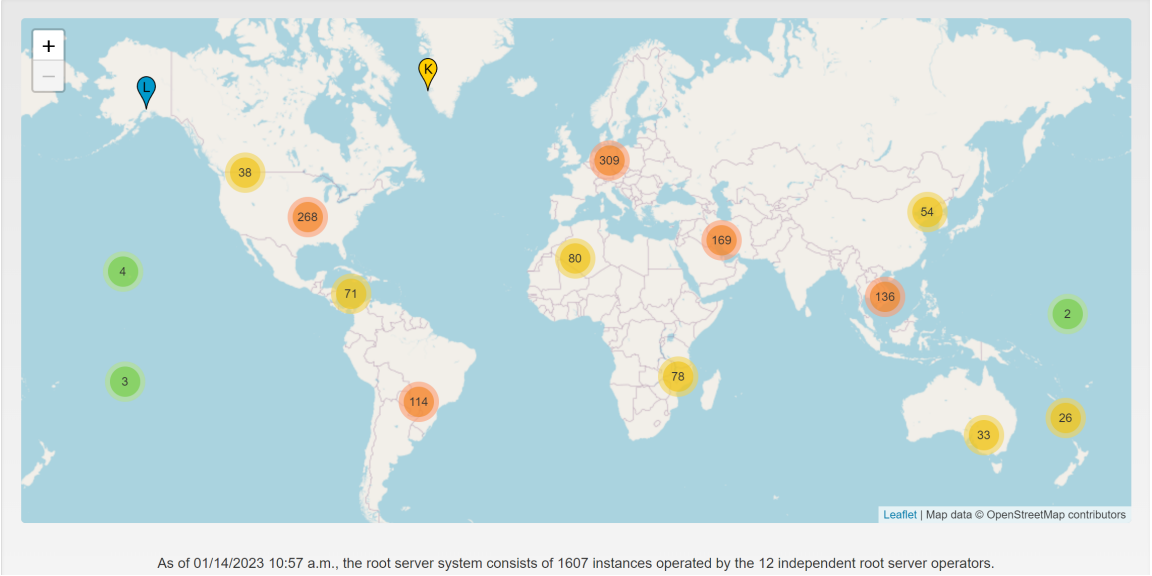


Figure 5.2: Root servers

As we research DNS queries for sensitive information we also want to handle this data appropriately. We had the following ethical considerations. Given a DNS query, there is potentially an end user or organisation behind the DNS query. In this thesis, we consider the privacy of the end user and organisation to be of importance. Information that can identify end users or organisations must therefore remain confidential. To achieve this the following was done:

- DNS datasets never left the servers of DNS-OARC or SURF.

- All analysis was performed on the servers of DNS-OARC and SURF.
- Exporting results was done with potentially identifying information anonymised.

From the DNS-OARC root server datasets, we have chosen datasets from the B-root, D-root and K-root. There was no particular reason for this and the datasets were chosen at random. A summary of the data can be found in table 5.1.

Root server	B-root			D-root	K-root
Server location	ams	ari	lax	amnl	ams
Number of queries	1786M	613M	812M	143M	410M
Total dataset	3764M queries				

Table 5.1: Number of queries from every DNS-OARC dataset

These datasets contain data captured on the days of 13, 14 and 15 April 2021. The B-root dataset was partially anonymised by the B-root operators. This was done by changing the lowest 8 bits in source IPv4 addresses and for IPv6 addresses, the lowest 32 bits were changed. The other two datasets from D-root and K-root have no adjustments to the original data. Another source of data comes from the SURF DNS resolver in Tilburg [30]. This data was collected from May 20th 2022 till 2nd June 2022 and contained 484 million queries.

DNS-OARC provided us with a machine to analyse the DNS data. SURF also provided us with a machine to analyse the SURF DNS data. Albeit not so powerful as the DNS-OARC one, but powerful enough nonetheless. Table 5.2 summarises the machines that were provided by DNS-OARC and SURF to analyse the DNS data.

	DNS-OARC setup	**SURF setup**
CPU	4x Intel Xeon X7560@2.27GHz	Intel Xeon Gold 5215@2.5GHz
RAM	144GB DDR3	4GB
OS	64-bit Debian Linux 8	Ubuntu 21.10

Table 5.2: Specification of both machines

All datasets were first preprocessed to extract relevant data from them. The DNS-OARC datasets were provided in a PCAP format and we used Tshark [31] to extract information from the PCAPs. We extracted the following information from every packet inside the PCAPs.

- Frame number.
- Source IP address.
- Destination IP address.
- Query name.
- Query type.

This was saved as one line in a `txt` file where every line represents a packet. A small snippet as an example is shown below:

Example DNS-OARC datasets after processing them

```
87134 136.228.26.254 199.7.91.13 wdziub.casinobank.nl 15
87135 128.225.26.124 199.7.91.13 <Root> 255
87136 146.161.245.240 199.7.91.13 macbook-huynguyen 1
87137 80.220.22.124 199.7.91.13 djiwjaeddoa 28
87138 118.34.87.90 199.7.91.13 radboud.nl 1
```

Note that the IP addresses used in this example and all upcoming ones are also random and not from real results.

The SURF DNS datasets were not PCAPs, but CSV files. An example of this is shown below.

Example SURF Tilburg resolver data.

```
2022-05-24T17:25.00.55794+02:00,radboud.nl,A
2022-05-24T19:25.00.53792+02:00,brightspace.com,A
2022-05-24T19:30.00.48059+02:00,surfnet.nl,A
2022-05-25T17:25.00.98285+02:00,gelderlander.nl,A
2022-05-24T17:25.00.98922+02:00,tweakers.net,A
```

This required less preprocessing and the only information we needed to extract from this was the QNAME. We do not have the source IP address in the SURF data as it would be the same for every query, namely the SURF resolver.

5.2 Data analysis techniques

In this section, we explain how the data analysis techniques were set up and used in this thesis. From used parameters to steps that were taken to analyze the data.

5.2.1 Regular expression filters

Regular expressions are used as filters to apply to the dataset. Using these regular expressions we can filter out the data that we are looking for. During our research, we searched for the following sensitive information in the datasets using a combination of regex filters and Python libraries.

- **Emails.**
  As emails can contain personally identifiable information we looked at emails that occurred in DNS queries.
- **Cities.**
  We look for cities in DNS queries as this can sometimes hint where a DNS query originates from.
- **Personal names.**
  Personal names might indicate who is behind a query.
- **Login patterns.**
  Potentially security sensitive information within DNS queries. For example, a protocol such as RTSP [32] (used for IP cameras) has a login URI [33] that looks like "rtsp://username:password".
- **IP addresses.**
  Sometimes DNS queries have an IP address in the QNAME. This can be unwanted information that leaks through a DNS query.
- **MAC addresses.**
  MAC addresses are 12-digit hexadecimal identifiers assigned to network interface devices such as routers, printers and phones. Finding MAC addresses in DNS queries can potentially reveal what kind of devices are being used in a network.

From a technical perspective, we use a tool called ripgrep [34] in combination with a regular expression to quickly go through the data. Ripgrep was used as it is extremely fast and this was needed when going through billions of DNS queries. An example of how this would look in the Linux terminal is:

Example regex

```
rg "radboud" -g 'dnsqueries.txt'
```

This command invokes "ripgrep" and tells ripgrep to search through the file "dnsqueries.txt" for queries that match "radboud". It then returns all DNS queries that have "radboud" in their QNAME.

However, there is a limitation when using regular expressions. As regular expressions are filters, there is no understanding of context. Regular expressions can return DNS queries that are unrelated to the main subject that we were looking for. Some examples of this:

- **Overlap with other categories.**
  Categories such as names and cities can overlap with each other. For example, Huy is a common Vietnamese name and a Belgium city at the same time. Other examples of overlapping categories are login patterns and emails. There are occasions where passwords in login patterns contain "@" and as a result, we also get login patterns when searching for emails of the format "abc@def". In general, there is no effective way beforehand to filter out these overlaps so it remains manual work afterwards to inspect the results.
- **Local IP addresses.**
  Because of NAT [35], IP addresses that we find in QNAMEs of DNS queries can be local IP addresses and not public ones. This does not always have to be a problem. For example, if we see the same local IP address occurring from the same subnetwork over and over we can potentially still identify that a certain device is active within that subnetwork.

5.2.2 Word2vec

We previously talked about how regular expressions can not understand the context of DNS queries. This is not a fault of regular expressions, as it was not intended for this purpose. This problem is somewhat alleviated by using the word2vec algorithm. This algorithm allows us to find associations between words that are being used in similar contexts. To understand why word2vec can be useful we first look at an example of syntactic similarity. For instance, a technique to find how similar two strings are is the Levenshtein distance. This is the number of insertions, deletions and substitutions that are needed to change one string into the other. The following example shows the Levenshtein distance between two names.

Example syntactic similarity with the Levenshtein distance

```
Thomas
Stephanie
Levenshtein distance = 7
```

A simple algorithm that is unfortunately not useful when applying it to DNS queries. In this example, the Levenshtein distance is relatively high at 7. Which is higher than the character length of "Thomas" at 6. This would indicate that these two words are not similar to each other. However, from a semantic perspective there is an association between "Thomas" and "Stephanie" as these are both first names. Using the Levenshtein distance would not help us recognise this association in this case. However, using word2vec we can potentially uncover the association between "Thomas" and "Stephanie" as these words are used in a similar context in DNS queries and will therefore be categorized as similar by word2vec.

Example first names

```
156 136.228.26.254 199.7.91.13 thomas-iphone 1
157 128.225.26.124 199.7.91.13 stephanie-macbook 255
158 136.228.26.25 199.7.91.13 thomas-macbook 1
159 128.225.26.124 199.7.91.13 stephanie-iphone 255
```

This example shows "Thomas" and "Stephanie" being used in a similar context. We see that both of these names are used as hostnames for a device. In this case, both "Thomas" and "Stephanie" are used as hostnames for their respective MacBook and iPhone. Word2vec would recognise this as both names are used in a similar location (first word) before the occurrence of "macbook" and "iphone".

To use word2vec we must first clean the data:

1. We can only process the QNAME of DNS queries so we remove everything else in the dataset that is not a QNAME.
2. We lowercase the QNAME to make sure that queries with different capitalisations such as "AMSTERDAM", "Amsterdam" or "amsterdam" are considered the same.
3. We remove the punctuation characters `!"#$%&'()*+,-./:;<=>?@[]^_`{|}~` and replace them with a space. This separates certain queries into multiple words and makes it easier for word2vec to analyse the relationship between them.

What remains is a cleaned-up dataset with words that can be fed into the word2vec algorithm. An example of this cleanup process is shown below.

Example dataset cleaning

```
156 136.228.26.254 199.7.91.13 destentor.nl 15
157 128.225.26.124 199.7.91.13 thomas-macbook 255
158 146.161.245.240 199.7.91.13 stefanie-macbook 1
159 80.220.22.124 199.7.91.13 RaDbOud.nl 1
```

becomes the following after cleaning

```
destentor nl
thomas macbook
stefanie macbook
radboud nl
```

However, we also lose some information with our method of dataset cleaning. During our second cleaning step, we lowercase everything. This causes words such as "Apple" representing a well-known tech company and "apple" as the fruit to be considered the same by word2vec. We accepted this flaw as lowercasing everything also brought us some benefits. It allowed us to have words that represent the same word but are differently capitalised to also be interpreted the same by word2vec. There are cases where this can be useful such as "Amsterdam" being capitalised in different ways yet they almost always referred to the capital of the Netherlands. Another problem with our method of dataset cleaning is that we have to deal with numbers in DNS queries. Think of IP addresses, small digits and MAC addresses. As these numbers do not often repeat themselves there is no proper association that can be calculated by word2vec. An approach was chosen to train the model in two ways. First, we would train one model without making any adjustments to any numerical value in the original data. Secondly, in some cases, we would replace these numbers with placeholders and train another model. For example, say that we have a big dataset and with some manual inspection we notice that there are DNS queries where smart camera's occurred together with IP addresses. We suspect that this is common and want to use word2vec to find out whether smart camera's and IP addresses indeed are associated with each other. We first perform our usual preprocessing as described above. We then replace all IP addresses in the dataset with the placeholder string "[IPv4]" to indicate that this used to be an IP address. Afterwards, we feed the data into word2vec.

Example replacing numbers with a placeholder

```
192.168.50.24-camera-john
192.168.50.25-smart-john
```

becomes

```
[IPv4] camera smart
[IPv4] smart john
```

As you see here the IPv4 addresses were replaced by the placeholder string "[IPv4]". This helps word2vec to associate "IPv4" with the words "camera" and "smart" as they occur in similar contexts.

Various parameters can be adjusted and tuned for word2vec. The following three were the most important ones for our models. With millions of queries as input, we had to ensure that word2vec did not consume beyond the processing and memory power of the machines we were using.

1. Vector size: 300. The authors of word2vec [9] point out that in general bigger vector sizes improve the accuracy of the final model. 300 was chosen as this value was also successfully tested by the original authors of word2vec. However, this was also chosen due to practical reasons. The bigger the number of dimensions, the more memory it would use. For reference, with a vector size of 300, we would roughly use up to 92GB of ram when training datasets of roughly 500M+ queries.

2. Window size: 5. This is the number of words around the trained word to take into consideration when training. 5 is the default value and a value that the original authors recommended when training on big datasets.

3. Type of model: Continuous bag of words (CBOW). There are two available models, namely CBOW and skip-gram. The difference between these two models is the input. CBOW takes the context (neighbour words) of the word it is training. Skip-gram uses the word it is given to predict the context around it. According to the original authors, CBOW works better on bigger datasets thus the choice was made to use CBOW.

An example of window size and CBOW vs skip-gram is shown below.

Example window size

www.simple.site.example.holiday.amsterdam.*com*

window size of 5

In this case, our input word is “example” and we have a window size of 5 around this word. These are the words that word2vec will consider when looking at the input word “example”.

Figure 5.3 from the authors of word2vec [36] also shows the difference between CBOW and skip-gram.

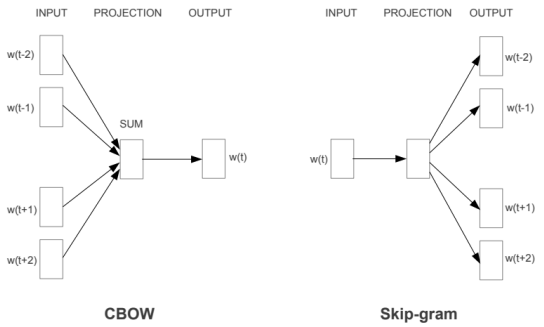


Figure 5.3: Different input between CBOW and skip-gram

After we give word2vec a dataset to train it will output a model where every trained word has a vector of size 300. We can use this model to find similarities between words in the dataset. This similarity is calculated as the cosine similarity of two vectors. As a use case example, say that we have a big dataset and with some initial manual inspection we spot some personal names. We want to know whether more names are occurring in the dataset. To do this, we can then use word2vec to create a model and help us detect other personal names that occur in the dataset.

Example searching similar words inside a word2vec model

```
model.most_similar('jan', topn=4)
```

$('piet', 0.983139), ('kees', 0.883539), ('jaap', 0.858123), ('henk', 0.841402)$

In this case, we searched for the name “jan” and we see as output other names that occur in the dataset. These are considered to be similar as they are being used in similar contexts.

5.2.3 Clustering

After filtering the DNS queries and creating a word2vec model our next step is to cluster the data. Having a word2vec model where we can find similar words is useful, but depending on the number of queries fed to the word2vec algorithm it is still quite a job to look through the data. Even a single day

of DNS queries at a single instance of a root server can contain up to 500 million queries. Clustering helps to reduce the amount of manual work that is needed when examining the data.

Two steps are needed to cluster the data properly. First of all, we need to pick the clustering algorithm that we want to use. The choice was made for mini-batch K-means. This was mostly a practical choice as this algorithm is fast and works well with large datasets.

Secondly, we need to determine the input for the clustering algorithm. Recall that the output of the word2vec algorithm is a model where we end up having vectors of 300 dimensions for every word in our dataset. Mini-batch K-means can work with vectors. So the choice was made to use the output of the word2vec algorithm as input for our clustering algorithm. However, the size of the vectors is too big to be clustered efficiently. We first need to reduce the 300 dimension vectors to 2 dimension vectors. We use UMAP for this. [37]. This is a dimension reduction algorithm. It has various parameters, but the most important ones are:

1. `n_neighbors` determines how many neighbours near a data point will be taken into consideration. This value was kept at the default value of 15.
2. `min_dist` controls the minimum distance between the points after dimension reduction. So in our case, it means the minimum distance of our 2 dimensional vectors after dimension reduction has been performed by UMAP. UMAP suggests a low value when clustering so the value of 0.0 was chosen. This is also the lowest possible value.
3. `n_components` determines the number of dimensions we want to reduce to. A value of 2 was chosen here as we reduce vectors of 300 dimensions into 2 dimensions.

To conclude, figure 5.4 shows the clustering process starting with DNS queries and ending with clusters.

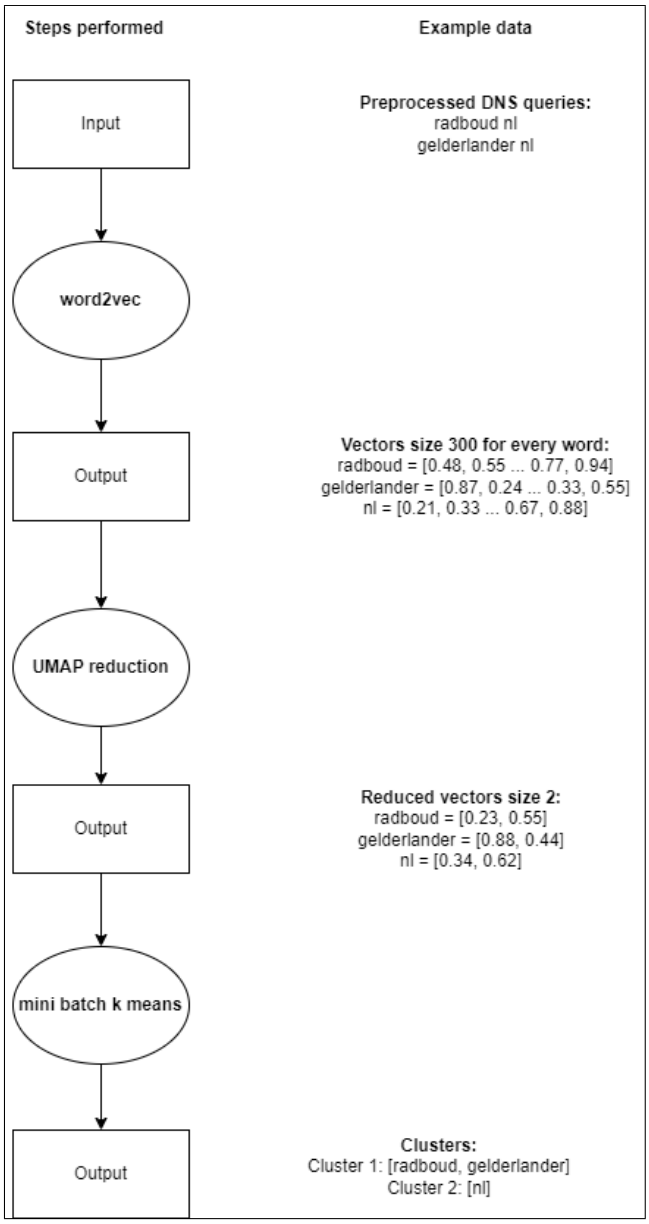


Figure 5.4: The clustering process

5.2.4 Entropy calculation of subdomains

There are applications out there that use subdomains for some form of user tracking. Thus it could be interesting to find these types of DNS queries. To find these queries we calculate the entropy of subdomains from the domains in the dataset. The reasoning behind this is that if we were to come across a domain with high entropy we might have stumbled upon a site that uses subdomains to do some form of tracking.

Example domain and subdomain

ywuwssasainc. degelderlander. nl
subdomain domain

To do this calculation we do the following:

1. We ignore domains in the dataset that do not have a subdomain.
2. Domains that have a subdomain are only considered if they occur more than 100 times in the dataset. This is to prevent domains that only occur a few times from showing up in our results. These domains are most likely not so interesting and usually have low entropy.
3. We collect the subdomains from every domain.
4. The probability distribution of the subdomains is calculated and saved in a list.
5. The entropy is then calculated over this list as $H(x) = -\sum_{i=1}^{n} P(x_i) \log P(x_i)$
6. The final result is the entropy value we are looking for.

Example subdomain calcuation

```
156 136.228.26.254 199.7.91.13 ywuwinc.destentor.nl 15
157 128.225.26.124 199.7.91.13 gelderlander.nl 255
158 146.161.245.240 199.7.91.13 gelderlander.nl 1
159 80.220.22.124 199.7.91.13 ywuwinc.destentor.nl 1
160 80.220.22.135 199.7.91.13 dkwokdw.destentor.nl 1
```

In this example, we see “gelderlander” and “destentor” as domains and only “destentor” has subdomains. Thus 2 unique subdomains from “destentor” will be used in the entropy calculation.

Following this entropy calculation on the dataset, we end up with a file with our entropy results that would look something like this.

Example entropy output

```
5.320391093 for degelderlander.nl
5.550395053 for amazonaws.com
6.302930193 for googleapis.com
8.320931032 for facebook.com
```

The first number indicates the entropy. The higher this number the more unique subdomains a domain has. So in this example, Facebook has the highest number and thus many unique subdomains. This begs the question of whether just counting the number of unique subdomains would have also sufficed. Counting the number of unique subdomains does not work because in some cases domains occur more often than others. For example, say that "degelderlander.nl" occurred 1,000 times while "facebook.com" occurred 1,000,000 times in the dataset. If "degelderlander.nl" had 900 unique subdomains this would be pretty high in entropy but low in numerical occurrences. If "facebook.com" had 901 unique subdomains this would be pretty low in terms of entropy but higher in numerical occurrences. However, 901 unique subdomains among 1,000,000 occurrences is not that high relatively speaking. Thus using entropy is easier to handle this difference. Following the results from the example, we could manually look at "facebook.com" queries to find out whether there are any queries that indicate some tracking in subdomains.

5.2.5 Linking IPs to an autonomous system (AS)

Autonomous systems (AS) are a group of IP networks [38] assigned to companies, ISPs, universities and various other organisations upon request. These autonomous systems are identified by a number and this is known as autonomous system numbers (ASN). Some examples of this:

Example autonomous systems

```
University of Cyprus, Cyprus, AS3268
SURF B.V., Netherlands, AS1103
Google LLC, United States, AS15169
Bol.com, Netherlands, AS199408
```

But what can we do with this information? For example, given the IP "145.116.16.128" we can find that it falls under the IP prefix "145.116.16.0/21". Given this prefix, we can then figure out that "SURF B.V., Netherlands" has announced this IP prefix. Thus seeing the IP address "145.116.16.128" in a DNS query tells us that this is a DNS query originating from the SURF network. However, we must consider that the SURF network serves many organisations in the Netherlands so it is hard to pinpoint exactly where this

query came from. But what if this DNS query came from a smaller entity? For example, say that we have a DNS query that is "samsung-router-E5413". If this DNS query were to come from an ISP that serves millions of users it is hard to pinpoint where this router is located. However, if the query came from a smaller AS such as a school or college it can be easier to narrow down where this query came from. Figure 5.5 shows the SURF example.

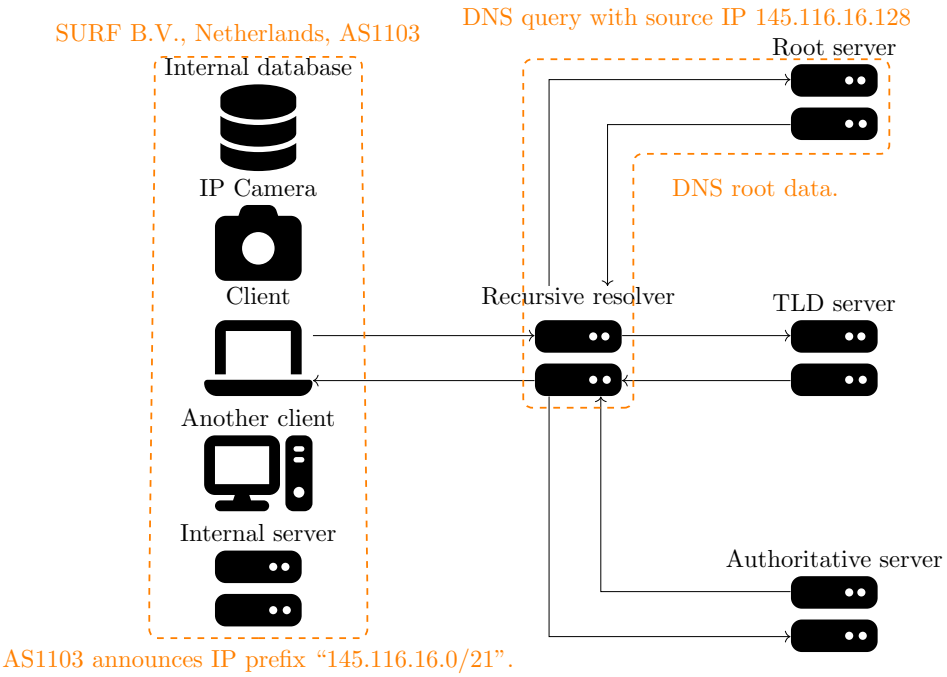


Figure 5.5: Example DNS query originating from SURF network

From an attacker's perspective, knowing what kind of software or hardware runs on a network makes it easier to find vulnerabilities.

Our step by step process to find the autonomous system name behind a DNS query:

1. Match the IP address of a query to the IP prefix it is part of.
2. Find the autonomous system number that announced this IP prefix.
3. Find who is assigned to this autonomous system number. For example, this could be a company, an educational organisation or an ISP.

The following example shows how this looks when applied to our dataset. We see that every IP address is preceded by the name of the autonomous system, the country and the autonomous system number.

Example of IPs being matched to autonomous system names.

```
GOOGLE, US 3339 87134 136.228.26.254 199.7.91.13 wdziub.casinobank.nl 15
TMOBILE, UK 5541 87135 128.225.26.124 199.7.91.13 <Root> 255
TMOBILE, NL 1554 87136 146.161.245.240 199.7.91.13 macbook-huynguyen 1
TTNET, TR 8282 87137 80.220.22.124 199.7.91.13 djiwjaeddoa 28
SURFNET, NL 9541 87138 118.34.87.90 199.7.91.13 radboud.nl 1
```

5.2.6 Presidio

Microsoft has developed a tool named Presidio that allows for sensitive data in unstructured text to be detected. Examples of this are IP addresses, credit card numbers, bank numbers, IBAN codes and various other recognisers. The goal is not only to find sensitive data but also to help anonymise it. The latter is less interesting for our thesis as we mostly focus on finding sensitive data and not anonymising it. Presidio works by using a combination of machine learning, regex, the context of the data and checksums to find various pieces of data that might be privacy or security sensitive. It also provides ways to extend the tool to add customisable recognisers. We used Presidio to help detect privacy and security sensitive DNS queries. Minor preprocessing is done as we only feed the QNAME of DNS queries to Presidio. Below is an example of the output of Presidio.

Example Presidio output

```
[Type:  IP_ADDRESS, start:0, end 9, score=0.5] 128.0.0.0
[Type:  URL, start:0, end 15, score=0.5] youtube.com
[Type:  LOCATION, start:0, end 14, score=0.8] london.uk
[Type:  IP_ADDRESS, start:0, end 15, score=0.8] 128.0.0.0:5447
```

We see that Presidio recognises information such as an IP address or a location such as London.

Chapter 6

Results

Our second sub research question was the following:

Sub 2 How can data analysis techniques help us find privacy and security sensitive information in DNS queries?

In the previous chapter, we described the data analysis techniques that we researched for this thesis. In this chapter, we go over the results. The results have been anonymised to not contain privacy or security sensitive information. This is highlighted by putting the anonymised parts between square brackets as follows:

> **Example anonymisation**
>
> `sensitive-database://huy-nguyen:somepassword123`
> becomes
> `sensitive-database://[username]:[password]`
> ———————————————————————————-
> Source: DNS-OARC datasets or SURF dataset

We go over some of the results and explain what techniques were used to find these results and why they are privacy or security sensitive.

6.1 Regular expression results

What we found: Login patterns for Real Time Streaming Protocol (RTSP) which can be used for IP cameras. The login URI for this is of the format "rtsp://username:password@ip".

Example RTSP

```
rtsp://ACCAdmin:[password]@192.168.16.241
rtsp://hydra:[password]@192.168.101.81
rtsp://admin:[password]@192.168.1.83:183
rtsp://deurbel:[password]@192.168.2.250:554
```

Source: DNS-OARC datasets

Why sensitive: In these examples, we see that the IP addresses are local addresses. We do obtain some username and password combinations that leak through DNS queries. In the last line we see "deurbel" which is Dutch for doorbell so this might also give away the location of a potential camera.

What we found: A non local IP address.

Example IP

```
http://[public IP]:7001/MagicInfo
```

Source: DNS-OARC datasets.

Why sensitive: Unlike the previous example where we saw local IP addresses, we have an example with a public IP address here. We found a Samsung MagicInfo [39] server which is used for various purposes such as monitoring, hardware control and remote login.

What we found: Username and password combinations from databases. An example of this is MongoDB which has "mongodb://user:password@ip:port/databasename" as login format.

Example MongoDB database queries

```
mongodb://appuser:[password]@localhost:27017/phhr
mongodb://127.0.0.1:27017/loyalty
```

Source: DNS-OARC datasets.

Why sensitive: In this case, the first line shows us a MongoDB connection with a username and a password. The second line tells us there is no username and password. Aside from leaking usernames and passwords, this also leaks that there is a MongoDB database running on the network.

What we found: Error messages from a database.

Example Oracle database error queries

1: clskec:has:CLSU:910 4 args[clsdAdrInit_CLSK_err] ...

Source: DNS-OARC datasets

Why Sensitive: As we mentioned in our technical background section, erroneous configurations resulting in invalid DNS queries can leak information. For this example, this error message tells us that an Oracle database is in use. This leaks what kind of software is being used in a network.

What we found: Error messages from monitoring software.

Example FourthShift monitoring software error queries

ERP XSMLSFZU46 - Task 64X for FourthShift 7.5F Report - Error Msg - Mail sending

---

Source: DNS-OARC datasets

Why Sensitive: Similar to the previous example, we once again have a DNS query that leaks what kind of software is being used. In this case, it is FourthShift 7.5 which is used for enterprise resource planning (ERP) and can be used to monitor critical business information [40].

What we found: Email addresses in DNS queries.

Example email addresses

`https://api-user.huami.com/registrations/[email]`

Source: DNS-OARC datasets

Why sensitive: In this case, we see what looks like Huami [41] smartwatches that are being registered. This is a potential privacy leak as emails can be personally identifiable information.

What we found: coordinates in DNS queries. Something we also came across were coordinates in DNS queries. In one particular case we found coordinates that point to a location in China and also included a username or domain name. This username or domain name also seems to link back to a user originating from China.

Example coordinates

https://user.[username/domain name].com_set_location.php?longitude=[coordinates]&latitude=[coordinates]

Source: DNS-OARC datasets

Why sensitive: Potential giveaway of location. Perhaps this is someone who is viewing this page or this is the person behind the actual page.

What we found: Cities, addresses and organisations in DNS queries. For example, we searched for Dutch cities in DNS queries.

Example cities in DNS queries

```
_ldap._tcp.[city]-[address]._sites.dc._msdcs.[organisation]
_kerberos._tcp.[city]-[address]._sites.dc._msdcs.[organisation]
_ldap._tcp.dc._msdcs.[organisation].local
```

Source: DNS-OARC datasets

Why sensitive: In these cases we have software information leaking from DNS queries. Including city, address and the organisation that is potentially behind these queries.

What we found: City, address and device combination.

Example smart devices in DNS queries

```
1212-AB3Sv19.[address].[city].zorgdomotica.shl.local r
```

Source: DNS-OARC datasets

Why sensitive: This query contains "zorgdomotica" which is a Dutch word that translates to smart devices. For example, safety sensors, cameras or emergency devices. All of these devices are usually automated and aimed to help the elderly. Think of situations such as forgetting to turn off the stove and a sensor that can help detect this. In this example, we have a DNS query that is potentially leaking that this kind of device is running on the network.

What we found: City, postal code and address combination. In this case, we found another Dutch city, but this time it also contains a Dutch postal code and address in the query.

Example cities in DNS queries

```
ces1-[postal code]-[address number].[city].nl.mon.local
ap1-[postal code]-[address number].[city].nl.mon.local
pes-[postal code]-[address number].[city].nl.mon.local
ap2-[postal code]-[address number].[city].nl.mon.local
```

Source: DNS-OARC datasets

Why sensitive: Similarly to the previous example, we have a DNS query with a location inside the query. This time we also have a postal code and the exact address number.

What we found: DNS queries with a date, timestamp and Linux version.

Example cities in DNS queries

```
_KERBEROS._udp.sles12sp3-SDO-automation-2021-01-12-18-30-26
_KERBEROS._udp.sles12sp3-SDO-automation-2021-11-12-15-30-45
_KERBEROS._udp.sles12sp3-SDO-automation-2021-11-08-13-30-22
_KERBEROS._udp.sles12sp3-SDO-automation-2021-09-25-12-30-36
_KERBEROS._udp.sles12sp3-SDO-automation-2021-11-10-18-30-57
```

Source: DNS-OARC datasets

Why sensitive: In this case "sles12sp3" refers to SUSE Linux Enterprise Server 12 SP3 and we have what looks like dates. As the dataset contained data collected on April 2021, some of these dates refer to future dates. It is not entirely clear what these dates refer to. However, by April 2021 this operating system was already outdated and end of life [42] unless additional support was purchased. This query highlights a potential security issue.

6.2 Results found with entropy calculation of subdomains

What we found: During our entropy calculations we came across a particular domain named "gansdorp". Gansdorp refers to a fictive place in Donald Duck. Unfortunately, this does not tell us much. However, the queries contained identifying laptop hardware along with personal names. They are of the format "device-username.gansdorp.nl".

Example entropy calculations

```
2022-05-20T22:46:28, Laptop-[name].gansdorp.nl,A
2022-05-20T22:48:28, Laptop-[name].gansdorp.nl,A
2022-05-21T09:05:22, Laptop-[name].gansdorp.nl,A
2022-05-22T10:01:11, Laptop-[name].gansdorp.nl,A
```

Source: SURF datasets

Why sensitive: This pattern of DNS queries would repeat itself over the days and indicate activity during weekdays with no activity on weekends. This could potentially indicate the working hours of the person behind this device. Multiple results with various other usernames similar to the examples were also found.

What we found: We described in the previous chapter how it could be useful to trace back a DNS query to the autonomous system it belongs to. By doing this, we were able to find what kind of software was running on the network of a Dutch organisation.

Example queries originating from a Dutch organisation

```
klogstash
centos-logstash
stgt-mongodb001
devgt-mongodb
sensuredis46-1
```

Source: DNS-OARC datasets

Why sensitive: From these examples we were already able to establish a few things. This organisation uses Logstash for log purposes, the Linux distribution CentOS, MongoDB databases and Sensu monitoring combined with Redis. These DNS queries tell us what kind of software this organisation is using. This can potentially be security sensitive and is also an example of institutional privacy. From an attacker's perspective, this would aid in initial reconnaissance to find vulnerable software or hardware to target. We talked about this previously and 5.5 summarises this concept.

6.3 Word2vec results

As mentioned previously, word2vec can help us find associations between words. There is a built-in function `most_similar` that given an input word shows us other words in the dataset that are considered to be similar.

What we found: Personal names. We asked word2vec to show us similar words to "Thomas" and as a result we were shown various other personal names that were identified in the dataset.

Word2vec example result with names

print(wv.most_similar("thomas", topn=10))
('stephanies', 0.7467923760414124), ('lele', 0,7356469631195068),
('vaihingen', 0.7289267182350159), ('manuels', 0.724760890007019),
('nicolas', 0.7244575619697571), ('kugkktl', 0.7226061224937439),
('karolinas', 0.7204602360725403), ('jacek', 0.7117319703102112),
('ryans', 0.7105698585510254), ('konstantin', 0.7076401710510254)

Another word2vec example result with names

print(wv.most_similar("andy", topn=10))
('liam', 0.8063532114028931), ('rafael', 0.7879030108451843),
('virdee', 0.7824928760528564), ('mostrador', 0.7773032784461975),
('morita', 0.7751322388648987), ('silvana', 0.7734751105308533),
('copia', 0.7732036113739014), ('keith', 0.7725027799606323),
('leos', 0.7641682624816895), ('oshfinance03', 0.763200044631958)

Why sensitive: Personal names are personally identifiable information. Being able to filter for names can be a first step in finding potentially more sensitive information related to these names.

What we found: In the next example, we look at the query "clashofclans". This is a popular mobile game. Word2vec returns us results with various other mobile games. Brawlstars, haydaygame, boombeach and clashroyal are all DNS queries referring to their respective mobile game. They are also all made by the same company. A deeper inspection of these DNS queries shows us that the DNS queries all have a similar structure and that is why word2vec sees them as similar.

Word2vec example result with android games

print(wv.most_similar("clashofclans", topn=10))
('brawlstars', 0.8758177161216736), ('haydaygame', 0.8559072613716125),
('boombeach', 0.7711310386657715), ('nflxext', 0.7378272414207458),
('qxwz', 0.7282516360282898), ('afcdn', 0.7265098094940186),
('clashroyale', 0.7232162952423096), ('gcloudsdk', 0.7211138606071472),
('seeds1', 0.7148250341415405), ('gamea', 0.7115039229393005)

Why sensitive: In this example we wanted to show what word2vec could recognise. These queries were not necessarily sensitive. Perhaps a small case

can be made for detecting user activity, but we wanted to highlight with this example what word2vec is able to associate.

What we found: This example with Facebook shows us keywords or similar queries that come from Facebook owned software. Upon inspection in the dataset, these queries were built up in a similar way in a similar context. Thus word2vec can recognise these queries as similar.

Word2vec example result with Facebook

print(wv.most_similar("facebook", topn=6))
('fbpigeon', 0.7020406723022461), ('instagram', 0.6789866089820862),
('accuweather', 0.6179394721984863), ('cdninsta', 0.6179394721984863),
('feeeek', 0.5753770470619202), ('googleapis', 0.5572503434753)

Why sensitive: We can identify the software that is being used within a network with some of these queries.

What we found: In this example, we have Apple devices that were considered to be similar. These DNS queries were most often in the form [apple device]-[username].

Word2vec example result with Apple devices

print(wv.most_similar("macbook", topn=10))
('ipad', 0.8398528695106506), ('imac', 0.8355129957199097),
('simone', 0.7761099338531494), ('air', 0.7725850939750671),
('urij', 0.7719191908836365), ('garys', 0.76851487159729),
('homepod', 0.7679030895233154), ('rabbit01', 0.7603201866149902),
('uti102', 0.7587342262268066), ('pauls', 0.7581713199615479)

Why sensitive: We can potentially detect the devices that are active within a network and also who they belong to.

What we found: Our last example for word2vec is an example with ROC. We suspect that this can refer to a Dutch word and roughly translated it is an abbreviation for regional education center. These centers are all over the country and in this particular example word2vec was able to find an association between ROC and the Dutch word Leijgraaf. Upon further inspection we can indeed find that there is an ROC named Leijgraaf.

Word2vec example result with ROC

print(wv.most_similar("roc", topn=10))
('leijgraaf', 0.7055112719535828), ('vmureswmdb01', 0.6953033804893494),
('dboit', 0.692886233329773), ('2k3', 0.692209005355835),
('vmurelay011', 0.683229386806488), ('vmu', 0.6823919415473938),
('wint', 0.6752498745918274), ('materialise', 0.6630422472953796),
('fvlprod', 0.6591544151306152), ('bulutu', 0.6581811308860779)

Why sensitive: This example was not immediately leaking sensitive information, but it can serve as a first step to find what kind of information is present inside a DNS dataset. For example, we searched for ROC to potentially find out whether a DNS dataset contains DNS queries originating from the Netherlands. In this case where we encounter "leijgraaf", we could for example continue our search filtering the DNS dataset for "roc leijgraaf" results.

6.4 Clustering results

It is impractical to list the clusters we get when clustering millions of queries. Figure 6.1 shows what this looks like.

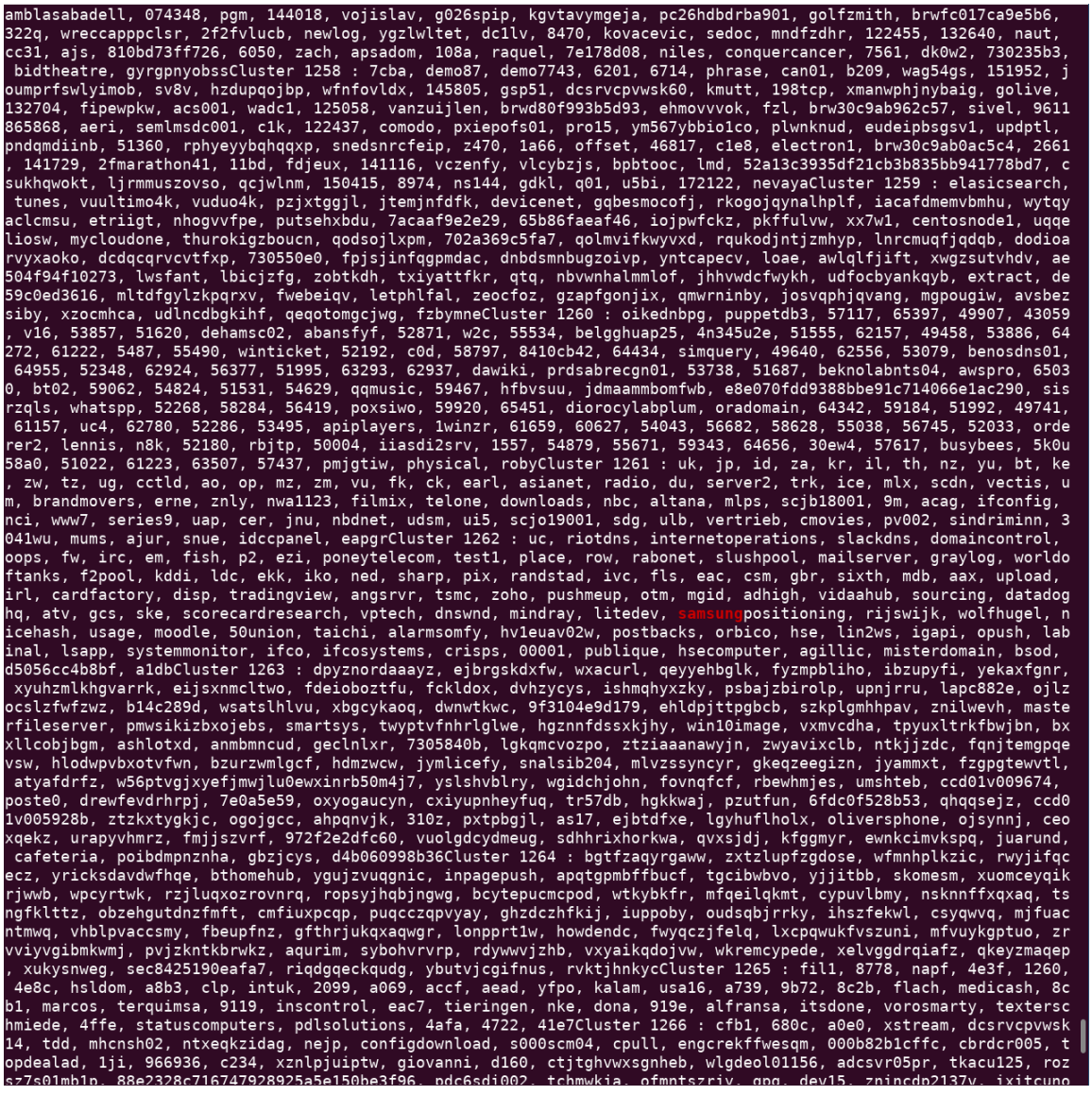


Figure 6.1: Clustering

Therefore it is more practical to highlight some results: In figure 6.2 we have a cluster where we highlighted two groups. One of them is a cluster with personal names and the other one is phone models.

In our second cluster example in figure 6.3 we have a group of software/hard-

Figure 6.2: Cluster example 1

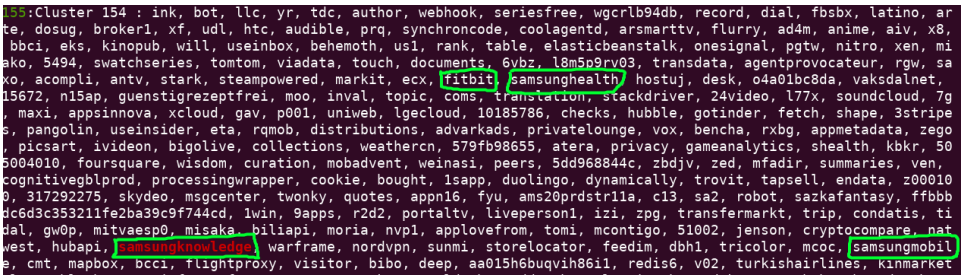


Figure 6.3: Cluster example 2

ware combinations. In this case, Fitbit is a smartwatch and Samsung software such as Samsung Health can be used with Fitbit to synchronize health data.

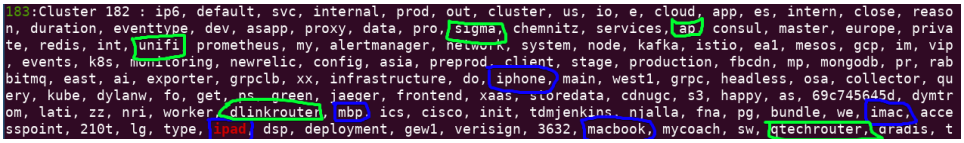


Figure 6.4: Cluster example 3

In our third example in figure 6.4 we have two groups. The first group is Apple devices such as iPhones, iMacs, iPads etc. The second group is routers. We see various brands of routers in this cluster.

Overall, clustering is an additional step to word2vec. It helps us to group DNS queries that are similar. As we see from the results it can help us find software, hardware and personal names.

6.5 Presidio

We were not able to find any interesting results with Presidio. Most of the findings of Presidio such as location, IP addresses or URLs were already found by using regular expressions. This is not completely unexpected as Presidio also uses regular expressions itself. The additional methods that Presidio uses to find information did not seem to provide additional interesting results when applied to DNS queries. There were also lots of false positives. For example, Presidio often saw some digits in DNS queries as a credit card number. As these digits were of a certain length, Presidio thought it matched some credit card number format. However, following our manual inspection, this did not seem to be correct.

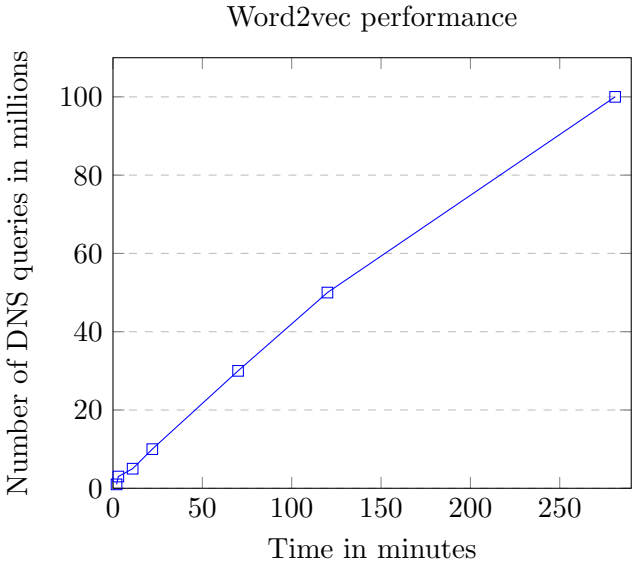
6.6 Performance

Given the size of our datasets, it is also important to go over some performance numbers. Having data analysis techniques that are also scalable will help us when researching bigger datasets. As a reminder these are the machines that we used during our research.

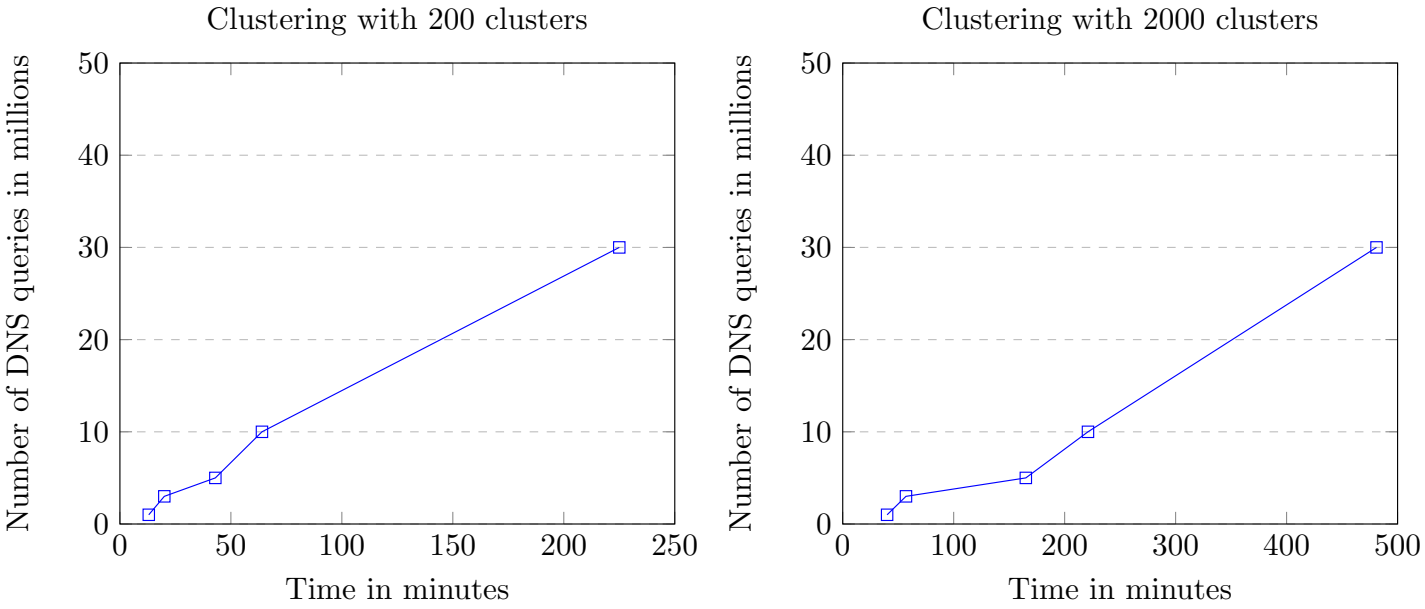
	DNS-OARC setup	**SURF setup**
CPU	4x Intel Xeon X7560@2.27GHz	Intel Xeon Gold 5215@2.5GHz
RAM	144GB DDR3	4GB
OS	64-bit Debian Linux 8	Ubuntu 21.10

Table 6.1: Specification of both machines

Let us first show some performance metrics of word2vec on the DNS-OARC machine. This was a machine shared with other researchers so not all resources were always available at all times. However, this gives an insight into the performances of word2vec.



We cluster using K-means and more specifically the Mini-batch K-means version from sklearn [43]. We show two examples. One where Mini-batch K-means was specified to create 200 clusters and another one with 2000 clusters.



A summary of the performances of the data analysis techniques is given in table 6.2.

Technique	**Input**	**Speed**	**Memory**
Regex filters	100M	<30min	<1GB
Word2vec	500M	48 hours	<90GB
Entropy calculation of subdomains	5M	45min-1 hour	<2GB
Clustering	See graph	See graph	<2GB
Presidio	100M	<30min	<2GB
Linking IPs to an autonomous system	<100M	<30min	<1GB

Table 6.2: High level overview performance of the data analysis techniques

We see that some methods such as regex filters are relatively fast and does not consume much processing power or take up much memory resources. On the other hand, methods such as word2vec or clustering take more time and resources. Thus, a realistic use case could be that we first manually

use regex filters to see whether a dataset contains some of the information that we are looking for. As regex filters are quick this does not take much time. Afterwards, we feed the dataset to word2vec so we can uncover more interesting queries. For example, if we are interested in personal names in a dataset we might first use a regex filter to quickly uncover whether the dataset contains personal names. We quickly check for the top 10 most popular first names. Moving on, we feed the full dataset to word2vec and word2vec can help us discover associations with personal names that we did not find initially.

Chapter 7

Discussion

In this section, we discuss our results, limitations of our work and potential future work.

7.1 Results

From our results, we see various cases where privacy or security information was leaked. However, it is still hard to verify how this happened. For example, we talked about how using entropy calculation on subdomains in section 6.2 we were able to find queries that contained a device name and personal name. However, it is unclear who is generating these DNS queries. Is it the device itself? Is it some other device on the network that is looking for this host? Is something wrongly configured somewhere in that network? There is no definitive answer possible from our side and we can only speculate. We know **something** is leaking that should not leak, as the root server cannot resolve a local hostname, but we do not know for certain **why** this is happening. In some other examples such as the postal codes and addresses example in section 6.1 it also remains unclear what is going on for similar reasons as mentioned above. In other cases such as the Oracle database example in section 6.1, we see a full error message in the query so we have more information about what is going on. However, it remains unknown why a full error message ends up on the root server.

When it comes to word2vec and clustering with K-means, we used these techniques to reduce the amount of manual work needed when searching for data in DNS datasets. However, we still have some manual work left mostly when reviewing the result. The results we get from Word2vec are vectors for every word that can be compared to other words in the dataset. For clustering, we get reasonably big-sized clusters. If we have 10M queries that are divided into 2000 clusters, every cluster will still contain 5000 results.

This is still quite some data to work through. It also depends on what we are looking for. Some cases are easier than others. For example, say that we are looking for all Samsung phones that occur in the dataset. This is a feasible task as Samsung phones have roughly the format "Samsung-[phone type]" in their DNS queries. However, what if we are interested in all the Linux distributions that occur in the dataset? In this case, we would have to know what these types of queries potentially look like. We showed examples of CentOS where the queries are "centos...", but we also saw that SUSE Linux had queries of the form "SLSE". Compared to our first task where we are looking mostly for "Samsung-[phone type]", the search for Linux distributions can be a harder task with more manual inspections.

An advantage of analysing DNS queries for sensitive information is that this is a passive undetectable method. Active analysis of a network with tools such as NMAP [44] or Shodan [44] are much nosier, detectable and can be blocked if detected. Passively analysing the DNS queries that go out of a network is not detectable. In some cases, we can also find information that is not possible with active analysis tools like NMAP and Shodan. For example, devices behind a properly configured firewall cannot be found by NMAP or Shodan. However, if they are generating DNS queries we might have a chance to detect them.

Understanding how privacy and security sensitive DNS datasets can be is also useful for research purposes. If certain DNS datasets indeed contain sensitive information, then we can choose to not share these datasets publicly with other researchers. Or perhaps we can anonymize the datasets if we know exactly which DNS queries contain sensitive data. Given our results, we have a recommendation when it comes to sharing DNS datasets. We know that it is possible to link DNS queries coming from certain IP ranges to a specific organisation. Thus second recommendation would be that if DNS datasets were to be shared, removing source IP from the dataset would prevent these kinds of linkages.

7.2 Limitations

One of the first data analysis techniques that we described was using regular expressions to filter out DNS queries. One of the issues with this is that regular expressions do not understand the context of DNS queries. Regular expressions are filters and they can only tell you whether a DNS query passed this filter or not. To remedy this lack of contextual understanding we tried another data analysis technique which relied on word embedding and word2vec. The word2vec algorithm takes an input word, looks at the words around it and compares it to other instances of the input word occurring in the dataset. This gives it a more contextual understanding of the data

which is a step in the right direction. With word2vec being used on DNS queries and the models that come out of it, we still have to wonder how accurate these word2vec models are. In more common use cases, word2vec is used to find word associations in text documents such as newspapers. Various datasets can be found trained on news stories from a wide variety of papers. Here, the accuracy of these models is checked by comparing them to the language that is being analysed. Say that we analyse various English newspapers using word2vec, then there are English datasets that have a good representation of the English language which can be used to validate the accuracy of the trained model. For example, one might have trained a new model where the results say that "Queen" is similar to "King". Or that "blue" is similar to "red". These results can be compared to an English dataset to confirm the accuracy of the results. However, when it comes to DNS queries things are a lot harder as there are no datasets available for comparison. How does one even conclude that queries are similar? There are no clear grammar rules in DNS queries when compared to the English language. There is no clearly defined structure when it comes to DNS queries. There might be some structure to certain DNS queries, but with lots of unique queries and random strings being used it is a lot harder to construct a dataset for comparison. The only way we validate word2vec models is by manually inspecting them. This is not the most ideal solution and certainly not a scalable one considering the size of the datasets. Aside from word2vec we also spent time clustering DNS queries. We would feed the output of word2vec as input for our clustering algorithm. This involved reducing the 300 dimensional vectors that represented a word to 2 dimensional vectors using UMAP. This remains a loss of data and can potentially affect the accuracy of the clustering results. In general, the values were chosen from a practical perspective. Given the sizes of the datasets, it was not always feasible to try many options for certain parameters. Aside from time constraints, it was also hard to compare results. As we mentioned previously, the accuracy of the results is hard to define and thus also hard to compare. However, this reduction still had to be done for practical reasons as clustering 300 dimensional vectors is a very intensive task. For clustering, the question remains the same as with word2vec. How do we know that these clusters are accurate? Unfortunately, just like with word2vec there is no clear answer to this. It remains a work of manual inspection and checking the clusters ourselves.

To conclude, optimizing parameters for the various algorithms used was a lesser concern in this thesis. We were more concerned with "can these data analysis techniques be useful to find the information that we want?" than "what are the most optimal parameters for our algorithms?".

7.3 Future work

During our research, we looked for login patterns using regular expressions and found results that resembled login patterns from various applications. From databases to IP cameras. In most cases, it looked like a wrong terminal command that managed to find its way to the root server. It can be interesting future work to uncover more login patterns that can potentially leak passwords and usernames.

As various media and IoT devices generate DNS queries for valid reasons there are also devices out there that deliberately send out DNS queries to domains that do not exist as a security measure. APNIC detected Chromium [8] browsers sending invalid DNS queries to the root server. As these queries will never be sent for a valid domain, it also means that they are never cached. This means that if these queries are done in regular intervals for security reasons you could potentially detect when these Chromium browsers are active. It might be interesting to look at this or other software with similar behavior.

We used some machine learning techniques when analysing the data such as word2vec, Presidio and clustering with K-means. Word2vec also has alternatives such as fastText [45] which is more focused on analysing subwords. Presidio also provides a framework to expand upon which can be interesting to look at. Other machine learning techniques such as natural language processing (NLP) can also be interesting to apply to DNS data. For clustering, there are other algorithms than K-means out there that can handle higher dimensional data. Recall that we used UMAP to reduce our 300 dimensional vectors to 2 dimensional vectors so we could perform K-means clustering. An algorithm such as HBDSCAN [46] claims to be able to handle higher dimensional data without too many sacrifices in performance. This can be interesting to explore as in our current clustering approach we reduce from 300 dimensions to 2 dimensions. Perhaps the results would be different or more accurate if we did not reduce this much.

Another point of interest is also researching how to accurately check the results of certain data analysis techniques performed on DNS queries. Our current method of manual inspections is not feasible with big datasets. However, perhaps it is still possible to check for a subset of the data or important queries in the dataset that should be grouped.

Chapter 8

Conclusion

During our research, we focused on understanding the privacy and security concerns of DNS queries. We also researched data analysis techniques to efficiently find sensitive information in large-scale DNS datasets. Our first sub research question was:

Sub 1 What type of information in DNS queries can be considered privacy or security sensitive?

We first looked at how a DNS query travels the internet. More often than not, DNS queries are unencrypted. Thus they can potentially be viewed by anyone. Afterwards, we had a look at two definitions of privacy for the context of our thesis. The first is from the GPDR. Which states that if any type of data can be traced back to a user, then it is privacy sensitive. In the context of DNS queries, this means that the information inside a DNS query is considered sensitive if we can use this information to find the user behind the DNS query. Another privacy definition came from Imana et al. [11] which is more aimed towards institutions. For example, can we use DNS queries to find out which institutions are communicating with each other? Following these definitions, we concluded that two types of information in DNS queries can be privacy or security sensitive. The first one is the source IP. This can potentially help us trace back who is behind a DNS query. Another type of information is the QNAME. This is the domain or information that is being requested by the DNS query.

Sub 2 How can data analysis techniques help us find privacy and security sensitive information in DNS queries?

To analyse over 4 billion DNS queries we explored the following data analysis techniques. Regular expressions were used as filters to apply to the DNS queries. This helped us to filter the data to look for information such as personal emails, login patterns and IP addresses. The downside of this is

that regular expressions do not have a contextual understanding of the DNS queries. To remedy this, we employed two machine learning techniques to help us. The first one was word2vec, an algorithm that recognises words that are being used in a similar context. This helped us to find associations between words in the dataset. The second one was clustering with mini-batch K-means. A clustering algorithm that allowed us to group the results found by word2vec. Other techniques such as calculating the entropy of subdomains allowed us to find domains that used their subdomains for tracking purposes. Another analysis method involved researching whether DNS queries originate from a specific place such as a company or university. The last data analysis tool we used was Presidio. A tool designed to identify personally identifiable information. All of these data analysis techniques helped us to efficiently and effectively find information in big DNS datasets.

Main What are the privacy and security concerns of DNS queries between the user and DNS servers?

Our privacy and security concerns were whether the information in DNS queries could affect personal end users or institutions. From our results, we saw various types of sensitive information in DNS data. DNS queries can leak login patterns and passwords. Or software level information such as certain databases running on a network. We also found DNS queries with locational data such as cities, addresses, coordinates and postal codes. Sometimes we also saw a device sending DNS queries periodically thus indicating activity coming from that device. To conclude, we have found privacy and security sensitive information in DNS queries sent between the user and DNS servers. However, not in all cases were we able to link the information in DNS queries to an end user or institution.

Bibliography

[1] "Domain names: Concepts and facilities," Internet Engineering Task Force, Request for Comments RFC 882, Nov. 1983, num Pages: 31. [Online]. Available: https://datatracker.ietf.org/doc/rfc882

[2] "Domain names: Implementation specification," Internet Engineering Task Force, Request for Comments RFC 883, Nov. 1983, num Pages: 74. [Online]. Available: https://datatracker.ietf.org/doc/rfc883

[3] "Domain names - concepts and facilities," Internet Engineering Task Force, Request for Comments RFC 1034, Nov. 1987, num Pages: 55. [Online]. Available: https://datatracker.ietf.org/doc/rfc1034

[4] "Domain names - implementation and specification," Internet Engineering Task Force, Request for Comments RFC 1035, Nov. 1987, num Pages: 55. [Online]. Available: https://datatracker.ietf.org/doc/rfc1035

[5] T. Wicinski, "DNS Privacy Considerations," Internet Engineering Task Force, Request for Comments RFC 9076, Jul. 2021, num Pages: 22. [Online]. Available: https://datatracker.ietf.org/doc/rfc9076

[6] A. Khormali, J. Park, H. Alasmary, A. Anwar, M. Saad, and D. Mohaisen, "Domain name system security and privacy: A contemporary survey," *Computer Networks*, vol. 185, p. 107699, Feb. 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128620313001

[7] J. M. Spring and C. L. Huth, "The Impact of Passive DNS Collection on End-User Privacy," *Securing and Trusting Internet Names.*, p. 11, 2012. [Online]. Available: https://www.semanticscholar.org/paper/The-Impact-of-Passive-DNS-Collection-on-End-user-Spring-Huth/4571cb7b246d9db40408c6f812861b1ea9f3bbfd

[8] M. Thomas, "Chromium's impact on root DNS traffic," Aug. 2020. [Online]. Available: https://blog.apnic.net/2020/08/21/chromiums-impact-on-root-dns-traffic/

[9] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," *Proceedings of the International Conference on Learning Representations*, Sep. 2013. [Online]. Available: http://arxiv.org/abs/1301.3781

[10] "2.3. Clustering." [Online]. Available: https://scikit-learn/stable/modules/clustering.html

[11] B. Imana, A. Korolova, and J. Heidemann, "Institutional privacy risks in sharing DNS data," in *Proceedings of the Applied Networking Research Workshop*, ser. ANRW '21. New York, NY, USA: Association for Computing Machinery, Jul. 2021, pp. 69–75. [Online]. Available: https://doi.org/10.1145/3472305.3472324

[12] "Content Taxonomy -." [Online]. Available: https://iabtechlab.com/standards/content-taxonomy/

[13] B. Imana, A. Korolova, and J. Heidemann, "Enumerating Privacy Leaks in DNS Data Collected above the Recursive," *NDSS: DNS Privacy Workshop.*, p. 7, 2018. [Online]. Available: https://www.isi.edu/~johnh/PAPERS/Imana18a.pdf

[14] W. Hardaker, "Analyzing and Mitigating Privacy with the DNS Root Service," *Proceedings of the ISOC NDSS Workshop on DNS Privacy*, p. 10, 2018. [Online]. Available: https://www.semanticscholar.org/paper/Analyzing-and-Mitigating-Privacy-with-the-DNS-Root-Hardaker/10670cd788a9eb31b6366a4c76d872cdf5824615

[15] A. Satoh, Y. Nakamura, D. Nobayashi, K. Sasai, G. Kitagata, and T. Ikenaga, "Clustering Malicious DNS Queries for Blacklist-Based Detection," *IEICE Transactions on Information and Systems*, vol. E102.D, no. 7, pp. 1404–1407, Jul. 2019. [Online]. Available: https://www.jstage.jst.go.jp/article/transinf/E102.D/7/E102.D_2018EDL8211/_article

[16] W. Lopez, J. Merlino, and P. Rodriguez-Bocca, "Vector representation of internet domain names using a word embedding technique," in *2017 XLIII Latin American Computer Conference (CLEI)*. Cordoba: IEEE, Sep. 2017, pp. 1–8. [Online]. Available: http://ieeexplore.ieee.org/document/8226415/

[17] "End of Service Notice." [Online]. Available: https://www.alexa.com/

[18] "Art. 4 GDPR – Definitions." [Online]. Available: https://gdpr-info.eu/art-4-gdpr/

[19] M. Nottingham, "The Internet is for End Users," Internet Engineering Task Force, Request for Comments RFC 8890, Aug. 2020, num Pages: 10. [Online]. Available: https://datatracker.ietf.org/doc/rfc8890

[20] "Do the data protection rules apply to data about a company?" [Online]. Available: https://ec.europa.eu/info/law/law-topic/data-protection/reform/rules-business-and-organisations/application-regulation/do-data-protection-rules-apply-data-about-company_en

[21] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, D. Wessels, and P. E. Hoffman, "Specification for DNS over Transport Layer Security (TLS)," Internet Engineering Task Force, Request for Comments RFC 7858, May 2016, num Pages: 19. [Online]. Available: https://datatracker.ietf.org/doc/rfc7858

[22] P. E. Hoffman and P. McManus, "DNS Queries over HTTPS (DoH)," Internet Engineering Task Force, Request for Comments RFC 8484, Oct. 2018, num Pages: 21. [Online]. Available: https://datatracker.ietf.org/doc/rfc8484

[23] R. S. Operators, "Statement on DNS Encryption," p. 1, Mar. 2021. [Online]. Available: https://root-servers.org/media/news/Statement_on_DNS_Encryption.pdf

[24] D. K. Gillmor, J. Salazar, and P. E. Hoffman, "Unilateral Opportunistic Deployment of Encrypted Recursive-to-Authoritative DNS," Internet Engineering Task Force, Internet Draft draft-ietf-dprive-unilateral-probing-02, Sep. 2022, num Pages: 27. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-dprive-unilateral-probing

[25] S. Rose, M. Larson, D. Massey, R. Austein, and R. Arends, "DNS Security Introduction and Requirements," Internet Engineering Task Force, Request for Comments RFC 4033, Mar. 2005, num Pages: 21. [Online]. Available: https://datatracker.ietf.org/doc/rfc4033

[26] D. Atkins and R. Austein, "Threat Analysis of the Domain Name System (DNS)," Internet Engineering Task Force, Request for Comments RFC 3833, Aug. 2004, num Pages: 16. [Online]. Available: https://datatracker.ietf.org/doc/rfc3833

[27] S. Bortzmeyer, R. Dolmans, and P. E. Hoffman, "DNS Query Name Minimisation to Improve Privacy," Internet Engineering Task Force, Request for Comments RFC 9156, Nov. 2021, num Pages: 11. [Online]. Available: https://datatracker.ietf.org/doc/rfc9156

[28] "DITL Traces and Analysis | DNS-OARC." [Online]. Available: https://www.dns-oarc.net/oarc/data/ditl

[29] "Root Server Technical Operations Association." [Online]. Available: https://root-servers.org/

[30] "DNS-resolvers | SURF.nl." [Online]. Available: https://www.surf.nl/en/surfdomeinen-register-and-manage-domain-names-yourself/dns-resolvers

[31] "Tshark | tshark.dev." [Online]. Available: https://tshark.dev/

[32] A. Rao, R. Lanphier, and H. Schulzrinne, "Real Time Streaming Protocol (RTSP)," Internet Engineering Task Force, Request for Comments RFC 2326, Apr. 1998, num Pages: 92. [Online]. Available: https://datatracker.ietf.org/doc/rfc2326

[33] T. Berners-Lee, R. T. Fielding, and L. M. Masinter, "Uniform Resource Identifier (URI): Generic Syntax," Internet Engineering Task Force, Request for Comments RFC 3986, Jan. 2005, num Pages: 61. [Online]. Available: https://datatracker.ietf.org/doc/rfc3986

[34] A. Gallant, "BurntSushi/ripgrep," Oct. 2022, original-date: 2016-03-11T02:02:33Z. [Online]. Available: https://github.com/BurntSushi/ripgrep

[35] M. Holdrege and P. Srisuresh, "IP Network Address Translator (NAT) Terminology and Considerations," Internet Engineering Task Force, Request for Comments RFC 2663, Aug. 1999, num Pages: 30. [Online]. Available: https://datatracker.ietf.org/doc/rfc2663

[36] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed Representations of Words and Phrases and their Compositionality," in *Advances in Neural Information Processing Systems*, vol. 26. Curran Associates, Inc., 2013. [Online]. Available: https://proceedings.neurips.cc/paper/2013/hash/9aa42b31882ec039965f3c4923ce901b-Abstract.html

[37] "UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction — umap 0.5 documentation." [Online]. Available: https://umap-learn.readthedocs.io/en/latest/

[38] N. 2019, "What is an AS Number?" [Online]. Available: https://www.ripe.net/manage-ips-and-asns/as-numbers/as-numbers

[39] S. D. Solutions, "MagicINFO™ 9 | Digital Signage Software Solutions." [Online]. Available: https://displaysolutions.samsung.com/solutions/signage-solution/magicinfo

[40] "Infor ERP accelerates productivity and agility." [Online]. Available: https://www.infor.com/solutions/erp

[41] " -." [Online]. Available: https://www.huami.com/

[42] "Product Support Lifecycle | SUSE." [Online]. Available: https://www.suse.com/lifecycle/

[43] "Clustering text documents using k-means." [Online]. Available: https://scikit-learn/stable/auto_examples/text/plot_document_clustering.html

[44] "Nmap: the Network Mapper - Free Security Scanner." [Online]. Available: https://nmap.org/

[45] "fastText." [Online]. Available: https://fasttext.cc/index.html

[46] "The hdbscan Clustering Library — hdbscan 0.8.1 documentation." [Online]. Available: https://hdbscan.readthedocs.io/en/latest/index.html