Analysis of DNS in cybersecurity

Patrik Hudák

Declaration

Hereby I declare, that this paper is my original authorial work, which I have worked out by my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Patrik Hudák

Advisor: Mgr. Vít Bukač, Ph.D.

Acknowledgement

Many thanks to my advisor Mgr. Vít Bukač, Ph.D. and my consultant RNDr. Václav Lorenc for their inputs and feedback during the writing of this thesis.

I would also like to express my appreciation to Martin Čarnogurský for his exceptional ideas and knowledge.

Last but not least, I would like to thank *Rapid7 Labs*, mainly Jon Hart who helped me understand some of the concepts presented in this work.

Abstract

This thesis presents multiple concepts around the DNS (Domain Name System) as it relates to cybersecurity. A publicly available Forward DNS dataset from Rapid7 is used as an underlying source of information. Primarily, techniques for domain correlation and subdomain takeover are addressed. Custom tools and scripts were developed as part of this thesis to demonstrate the theoretical concepts on the aforementioned dataset.

Keywords

Contents

1 Introduction

With the adoption of DevOps and containerization, organizations are more and more switching to rapid prototyping and fast release cycles of applications. Whereas applications were previously hosted on-premise, today, cloud providers are becoming more dominant [Wei17]. Tools such as *Ansible*, *Chef*, or *Terraform* were created to help administrators efficiently manage resources, both in cloud and on-premise. These tools can automate the processes of creating a new server, setting up domain names, etc.

Such a transition is introducing new security concerns. By having a large digital footprint [1], organizations are far more exposed on the Internet. Cyber adversaries are trying to identify unpatched applications and services which can lead to initial exploitation, and, thus gaining access to internal networks. Another problem area is the management of domain names. Since large organizations are spread across the globe, the lack of centralized management of domain names often yields to various kinds of problems. Recall the security incident, which happened in 2012, when cyber adversaries exposed a significant number of *Yahoo* login credentials [Goo12]. The cause of this behind this breach was a vulnerable web application on one of Yahoo's subdomains where SQL injection was possible. It is not unusual to see publicly exposed applications and services which are not meant to be public [HHH17].

This thesis aims to shed light on techniques that cyber adversaries use to map, and possibly exploit, the digital footprint of their targets using domain names. Multiple DNS related research papers use only a limited set of domain names to demonstrate the concepts presented in the paper [LHW16] [Kha+15] [KKvE16]. This thesis provides findings and practical ideas, and demonstrates them using the extended domain name dataset. Organisations can use these ideas to understand their public exposure from the perspective of cyber adversaries. For some parts of this research, widespread misconfigurations were identified across the Internet. In fact, section 5.3 presents such misconfigurations (which are exploitable) being found in multiple high-profile organizations. These organizations were proactively notified so they could fix the discovered problems.

The remainder of this thesis is structured as follows. In *chapter 2*, fundamentals of the *Domain Name System* are covered. The process of finding a domain name dataset is presented in *chapter 3*. The topic of *domain correlation* is discussed in *chapter 4*. In *chapter 5*, concepts of *subdomain takeover* are presented along with results obtained during Internet-wide scans. Finally, *chapter 6* details conclusions.

---

1. Domain names, public facing hosts, etc.

2 DNS

Before explaining more advanced concepts and analyses, it is useful to describe fundamental properties of DNS. The DNS acronym stands for *Domain Name System*. It is an essential part of Internet Protocol (IP) suite. Virtually every application level (in ISO/OSI model) protocol requires IP address (IPv4 or IPv6) to establish a connection to the other party. Because people do not easily remember IP addresses, human-readable labels called *domain names* were created. Although DNS is often seen only as a protocol, it describes a complete system for manipulation of domain names. The next sections explain the parts of DNS, that will be used in the following chapters.

2.1 Domain Name Space

The primary purpose of DNS is to provide a domain name resolution, i.e., to translate domain names to IP addresses. Domain names are used for identification of network resources which are unique across different networks, protocol families, internets, and administrative organizations. DNS uses a hierarchy to manage its distributed system of domain names [DR17]. The DNS hierarchy, also called the *domain name space*, is a tree structure. The root node of this tree is represented by a root domain labelled as a "." (dot). Below the root domain are the top-level domains (e.g., *.com*, *.net*, *.info*, etc.) that further divide the DNS hierarchy. Below top-level domains are second-level domains and so on. Figure 2.1 illustrates the domain name space.
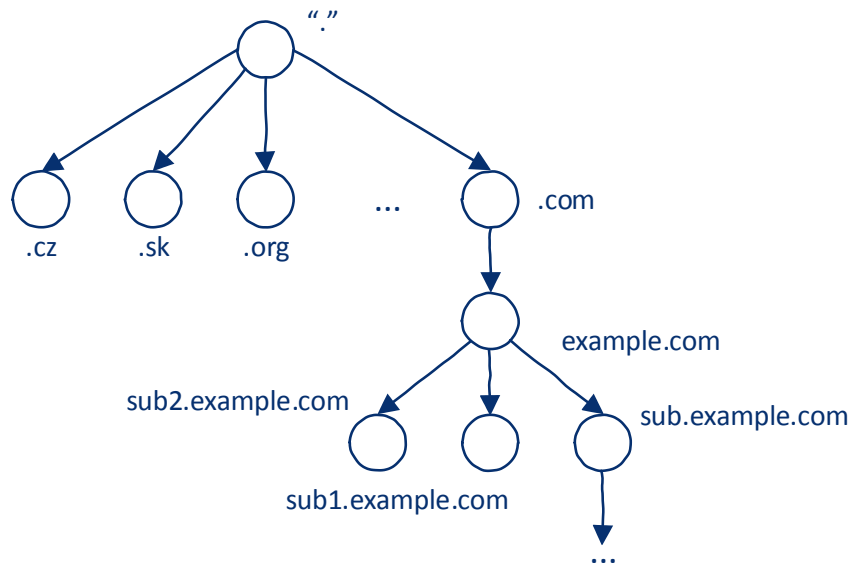

Figure 2.1: An illustration of domain name space. Each node on the DNS tree represents a domain. Under the top-level domains are domains of specific organizations.

A domain name is used to describe the position of the domain in the DNS tree. A domain name consists of one or more parts (also called *labels*) delimited by a "." (dot). Each of these labels represents one level on the DNS tree. The figure shows the breakdown of these parts.
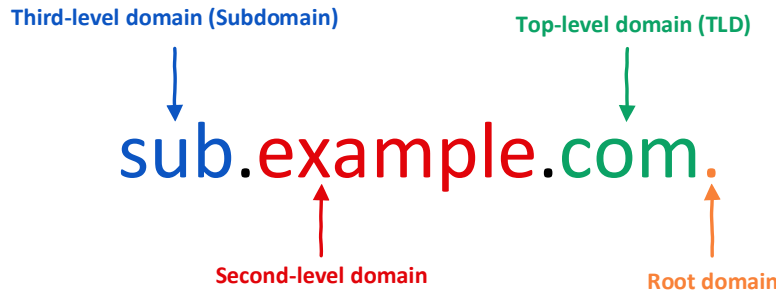

Figure 2.2: The illustration represents different parts (*labels*) of the domain name. The hierarchy of domains descends from right to left. Note that the root domain (final ".") is usually omitted in regular domain name representation.

Domain registration is a process of creating a new domain under one of the top-level domains (TLD). Domain registrar is an organization which can create new second-level domains. Note that there are multiple domain registrars (e.g., *GoDaddy*, *Namecheap*, etc). However, not all domain registrars can register domains under all TLDs. For instance, a new domain name under *.int* TLD must be explicitly approved by *IANA* [Bry16].

There are also domains, such as *co.uk*, which are second-level domains by definition, however, a domain registrar is required to create a „child" under this domain. Mozilla created a *Public Suffix List* that collects all TLDs and second-level domains which need a domain registrar to create new „children" of this domain [Moz]. To avoid ambiguity, the rest of this thesis will refer to the domain name that has only one label and suffix from the Public Suffix List as a **base domain**. The list of examples follows:

- Domain: *example.com*; Base domain: *example.com*
- Domain: *sub.example.com*; Base domain: *example.com*
- Domain: *example.co.uk*; Base domain: *example.co.uk*
- Domain: *sub.sub.example.co.uk*; Base domain: *example.co.uk*

2.2 Zones

A *DNS zone* is a portion of domain name space which is administrated by a single entity (a group of DNS servers). A DNS tree is divided into multiple administrative parts using DNS zones. One DNS zone manages information for one domain and any part of its subtree. The zone information is stored in a file called the *zone file*. This is a text file which contains

information such as domain to IP address mapping and much more (see section 2.3). Figure 2.3 illustrates the different DNS zones that can be formed on DNS tree.
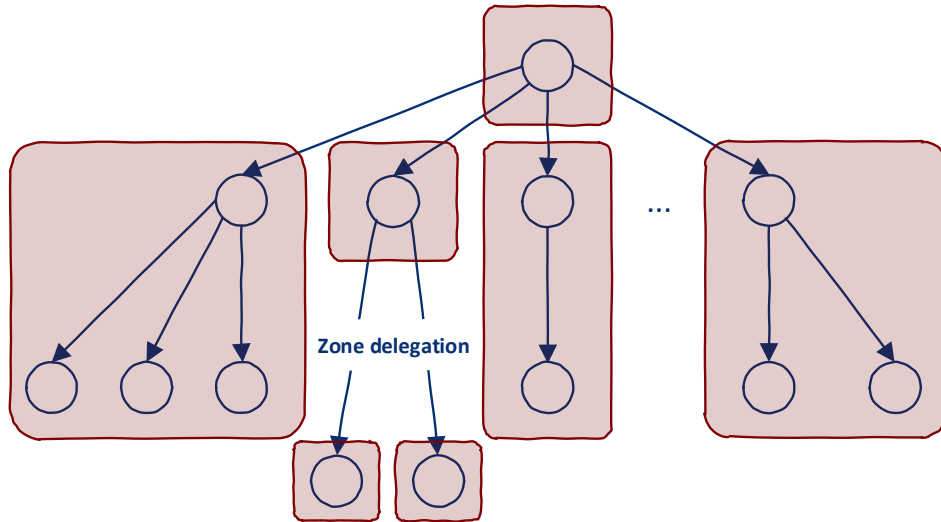

Figure 2.3: Examples of zones in domain name space. Nodes represent domains in the DNS tree, and red boxes represent DNS zones.

A DNS zone can delegate the portion of one of the subtrees it manages to another entity. This process is called *zone delegation* and is achieved using NS record (DNS records are explained in section 2.3). For instance, a DNS zone which has *example.com* under administration, can delegate the *subdomain.example.com* (and its whole subtree) to another DNS zone.

2.3   Resource Records

A zone file is a collection of resource records, with each of the records providing information about a specific object [LS16]. The standard resource record (RR) consists of five fields:

- **Name**
  Specifies the object that owns the particular RR. Depending on the record, the *Name* field is either the hostname or the IP address.

- **TTL**
  Time-to-live (TTL) defines the duration, in seconds, that a record may be cached. Zero indicates the record should not be cached. If no TTL is specified, then the default zone TTL value is used.

- **Class**
  Defines the constituency of the record. It is always set to *IN* (Internet), which is the only class currently supported by DNS.

- **Type**
  Specified semantic value or purpose of the record. The RR types are further explained below.

- **Data**
  The *Data* field holds the information for the current object. The value of the *Data* field depends on the RR type. For instance, in domain to IP address mapping, *Data* field is specified by IP address.

The example RR can look like this (the fields are delimited by a whitespace in the order as listed above):

```
example.com. 3600 IN A 93.184.216.34
```

Resource Record Set (RRset) is a name given to all records of the same type for a given domain. The list of basic records types is as follows:

- **SOA**
  Start of authority. The SOA defines global parameters (owner name, serial number, expiration dates, etc.) for the domain. Only one SOA record is allowed in single zone file and it must be the first RR specified in the zone.

- **NS**
  Name server. It is used for zone delegation. Usually, multiple NS records are present in every zone file.

- **A**
  Address. It holds the 32-bit IPv4 address. This is the primary type used for mapping from domain name to address.

- **AAAA**
  IPv6 Address. It holds the 128-bit IPv6 address.

- **CNAME**
  Canonical name. Alias of one domain name to another. CNAME can be seen as *"DNS redirect"*. This is useful in situations where there are a lot of other domain names pointing to the same IP address. When a change is made to the primary address, the others are updated automatically. CNAME records are heavily discussed in *chapter 5*.

- **MX**
  Mail exchange. Provides mail servers for a given domain. *Data* field also contains *record weight* which is unique to MX records. The weight determines preference for multiple MX records.

- **TXT**
  Text. Can hold arbitrary text data. Use cases of TXT records include *SPF*, *DKIM*, or *DMARC* configuration, which are used to prevent email spoofing.

- **PTR**
  Pointer. PTR records are mostly used for reverse DNS resolution. They are the opposite of A records: PTR records map IP addresses to domain names.

2.4 DNS Resolution

DNS protocol is a *client-server protocol.* DNS servers hold zone files and provide the contents to DNS clients. Zones are usually administered by two DNS servers called primary and secondary. The secondary DNS server acts as a backup mechanism in the case the primary DNS server fails. The primary DNS server holds the master copy of the zone file. The secondary DNS server periodically requests a copy of the zone file from a primary server using *zone transfer.*

In the context of the particular domain name, a name server is called *authoritative* if it manages the zone of the particular domain. For instance, if the zone of *example.com* is managed by *ns1.example.com* and *ns2.example.com*, then both these servers are authoritative for *example.com.* Any other DNS server is called *non-authoritative* for *example.com.*

If the DNS client wants to get the IP address (or other information from DNS zone) for a given domain, it needs to issue a DNS query and properly follow the zone delegation from the root domain. This process is called *DNS resolution.* Figure 2.4 illustrates DNS resolution for *example.com* domain.
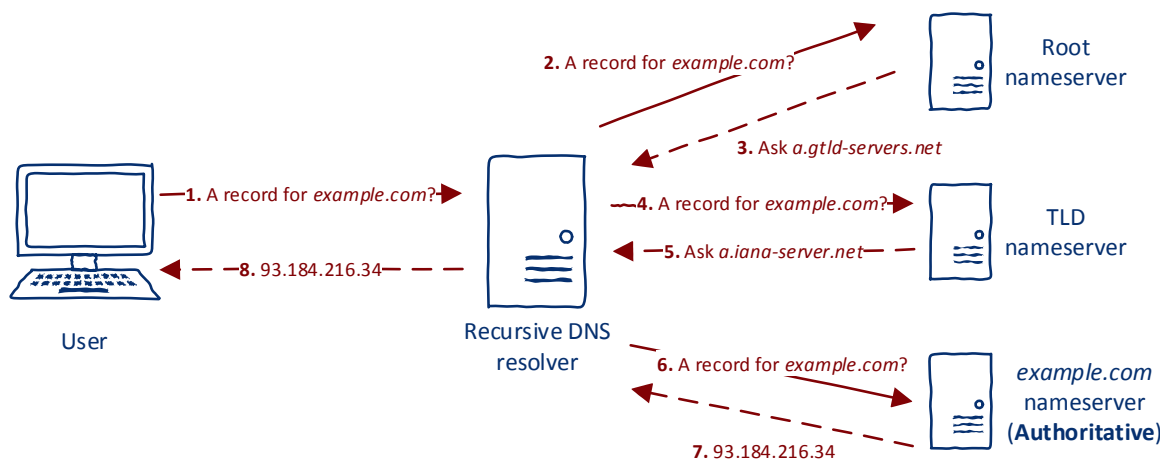

Figure 2.4: DNS resolution process. Notice that the recursive DNS resolver is communicating with other DNS servers on behalf of the user. Step #7 is the only authoritative response in the whole diagram. Although step #8 returns the correct A record, this response is non-authoritative because it is returned from non-authoritative DNS server (in context of *example.com*).

Note that in Figure 2.4, the user is not directly doing DNS resolution on her host, but rather uses an external server called *recursive DNS resolver.* The main reason behind this is caching. If more users share one DNS resolver, it can deliver results more efficiently because most popular resource records are already cached. Therefore, the DNS resolver does not perform the full resolution process which often consists of multiple queries to find the authoritative DNS server.

In addition to regular resource types, explained in section 2.3, there also exists so-called *pseudo-record types* or *meta-query types*. The difference is that meta-query types are not directly included in the zone file but are, rather, dynamically created by a DNS query. Not all meta-query types are properly documented, however, there are at least two widely used [DR17]:

- **AXFR**
  As mentioned above, the secondary DNS server requests the current copy of the zone file from the primary DNS server using zone transfer. The AXFR meta-query type is used for zone transfers.

- **ANY**
  ANY meta-query returns every resource record where *Name* field is the given domain name. The DNS server returns the subset of the zone based on the domain in the DNS query.

3 Dataset

The rest of this thesis is aimed towards the analysis of domain names and their properties from the cybersecurity standpoint. Such analysis requires some dataset from which the facts can be extracted. This chapter describes the process that took place while choosing the suitable dataset, its features, and statistics.
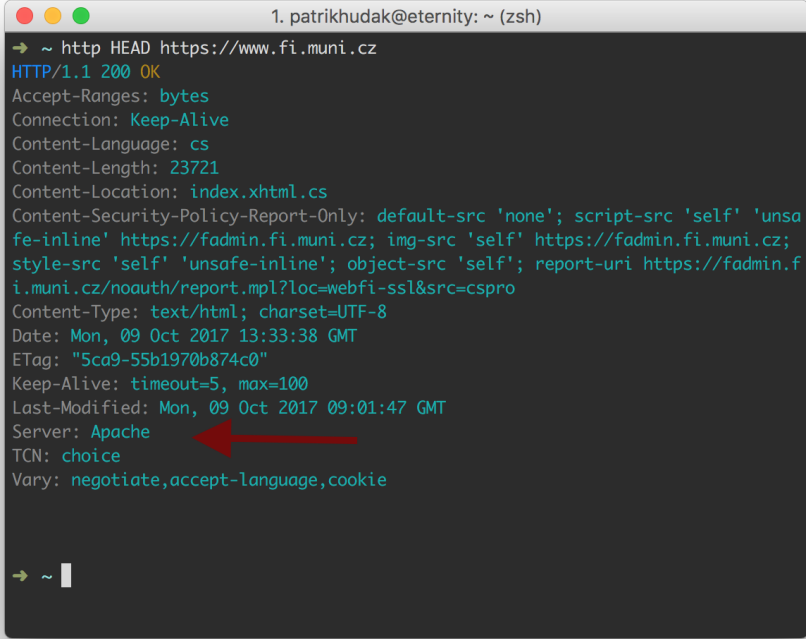
3.1 Internet-wide scans

Internet-wide scanning is a process of identifying publicly exposed services across the whole IPv4 space [DBH14]. The idea behind these scans is to have visibility of the hosts on the Internet. With the data obtained using Internet-wide scans, researchers can measure protocol adoption, identify large-scale problems, and much more. This chapter focuses on three publicly available projects related to Internet-wide scans: *Project Censys*, *Shodan*, and *scans.io*.

Project Censys is periodically scanning IPv4 space for well-known ports (e.g., 22, 80, 443). It tries to locate and identify publicly exposed services. The collected data is then freely accessible via REST API, web user interface, tables on Google BigQuery, and downloadable datasets [Dur+15]. Censys can also be seen as a form of search engine for Internet hosts. It can, for instance, list every publicly available web server running Apache, located in the Czech Republic. Figure B.2 presents a screenshot of Censys web interface.

Banner grabbing is a technique used to determine the type and version of software used on the scanned port [MSK12]. A banner is usually the first response received from the server after initial handshake. HTTP response headers are used as a banner for HTTP. HTTP response headers often contain *Server* field from which the web server information can be parsed. Figure 3.1 demonstrates this behavior. Banner grabbing can be extended to services like SSH, FTP, and others. Censys uses banner grabbing for producing additional information (e.g., software name and version) about service running on the scanned port.

This behavior is very comparable to another search engine called *Shodan* [Sho]. Similar to Censys, Shodan is doing periodic scans of IPv4 space. However, it is more optimized for the *Internet of Things (IoT)* devices. Shodan tries to bring awareness by providing a user-friendly web interface - the interface displays the view of the exposed web camera or screenshot of the open VNC server. Shodan tries to identify the services that still have common vulnerabilities, such as *Heartbleed*. On-demand scanning is also possible using Shodan. This feature is aimed mainly towards larger enterprises. Figure B.1 shows a screenshot of Shodan's web interface.

These two projects gained popularity in the past years, due to public reports and tweets from security researchers. Researchers revealed all kinds of security problems, mainly in the emerging IoT sector. There are countless examples of unsecured web cameras, devices controlling critical infrastructure, HVAC, or even power systems in large industrial organizations [HHH17]. Internet-wide scans help security researchers find such exposed devices, or devices containing well-known vulnerabilities. On the other side of the spectrum, cyber

Figure 3.1: The simplified example of banner grabbing. HTTP response headers from *fi.muni.cz* contain *Server* field with a value *Apache*.

adversaries are using Internet-wide scans to discover exposed assets, which might be compromised and used for malicious purposes. *Mirai* botnet, one of the largest botnets ever created, leveraged exposed IoT devices [Ant+17]. Mirai authors used Internet-wide scans to locate IoT devices with none or default password configuration. Such devices were then exploited and added to the botnet. In particular, devices with exposed *telnet* were primarily targeted. Another example is ransomware attacks against exposed MongoDB services. In early 2017, thousands of unauthenticated MongoDB databases were encrypted using ransomware [Bre17].

Alongside Censys, The Internet-Wide Scan Data Repository called *scans.io* was created [Cen]. Censys team maintains it at the University of Michigan. This repository contains raw data (datasets) gathered during Censys scans. Researchers can download the data directly and start doing experiments or calculate statistics, which would not be possible using only REST API or user interface of Censys. These datasets are offered without any cost.

Censys is using two essential components during their scans:

- **ZMap** [1]
  A fast single packet network scanner, which can scan the entire IPv4 under 45 minutes (with a gigabit Ethernet connection).

------

1. https://github.com/zmap/zmap

- **ZGrab** [2]
  A banner grabbing tool for the results provided from ZMap.

The datasets are split based on the port they address. Every dataset is divided into smaller parts, containing separate output from ZMap and ZGrab. The datasets are compressed using LZ4, and their inner format is usually JSON or CSV.

One of the datasets provided by scans.io is DNS (port 53) dataset. Since this is port 53 scan, it does not contain sufficient data related to domain names. This dataset contains DNS results from the transport layer, which, in fact, reveal only available DNS servers and resolvers. Unfortunately, this data is unusable for topics addressed in this thesis due to insufficient context.

3.2 Project Sonar

Luckily, scans.io repository is not limited to datasets from Censys. Other researchers and organizations share their researches in scans.io repository. *Rapid7*, a well-known cybersecurity company, runs their internal Internet-wide scans. Periodically (every seven days), they share the results of this research into scans.io. Rapid7 names this research *Project Sonar*. It is a name for their DNS, SSL, HTTP, and UDP scans.

Their approach in doing DNS scans is significantly different from what Censys is doing. DNS dataset in Project Sonar approaches DNS scanning from domain name perspective. *Forward DNS scan* starts with a large list of domain names, including higher-level domains. Sources for these domains are provided by [Har17]:

- **Dumps of Top Level Domain (TLD) zones**
  Some TLD operators allow requesting a full dump of their DNS zones. For instance, *Verisign* is an operator of .COM and .NET. Verisign allows downloading the dumps after submitting an official request [Wri15]. Dumps of the zone files are incredibly useful to provide base visibility. However, they often do not contain higher-level domain names, such as subdomains. It is not officially stated which exact TLD zones are retrieved by Rapid7. However, the list includes updated .COM, .INFO, .ORG, .NET,. BIZ, and .INFO zones.

- **SubjectAltName in SSL certificates**
  Parallel to DNS scans, Project Sonar scans port 443 and parses the information from SSL certificates. SSL certificates (X.509 certificates) contain extension called *SubjectAltName*. SubjectAltName allows specifying more domain names into one certificate. This technique is often used in virtual hosting configurations (see subsection 4.2.1) or with shared certificates. Domain names from SubjectAltName field provide many domain names that would not be found using only TLD zone dumps. However, there is a slight problem in scanning only IPv4 space for certificates. Due to the extended use of virtual hosting setup (described in subsection 4.2.1), TLS extension called *Server Name Indication (SNI)* was created [Bla+06]. Its main concept is this: If there are multiple websites hosted on the same IPv4 host, they can

---

2. https://github.com/zmap/zgrab

use different SSL certificates based on the domain that the client is requesting. In other words, if host 1.2.3.4 is hosting domains example.com and example.net, the domains can have separate certificates generated. The correct certificate is provided to the user based on a requested domain, which is communicated as part of TLS negotiation process. This domain name is sent unencrypted. Project Sonar has big blindspots in certificates because of not using SNI, but rather scanning IPv4 using only IP addresses. This means that the server might respond with zero or one certificate (based on its configuration). However, the server might be hosting hundreds other certificates (because of SNI and virtual hosting) that are not included in the Project Sonar dataset. On the other side, if the server is using virtual hosting setup without SNI, i.e. all websites are sharing one common certificate with SubjectAlt-Name setup correctly, all these domains will be extracted from the certificate. In addition to SSL certificates from port 443, Project Sonar also gathers certificates from other SSL/TLS services, such as IMAP, POP3, and SMTP [Moo15]. These certificates might contain other domain names, which would not be revealed using only port 443 scans.

- **Domain names from HTTP responses**
  Project Sonar also includes a scan of HTTP servers across IPv4 space. Additional domain names are extracted from HTTP responses. The places of these domain names include the *Location* field in HTTP response headers or just regular HTML elements such as anchor links or images.

- **PTR records**
  In addition to forward DNS scans, Rapid7 is also doing reverse DNS scans as part of Project Sonar [Mo015]. The reverse DNS dataset includes the responses to the PTR lookups across IPv4 address space. Domains obtained using PTR records are fed back to domain list used for forward DNS scans.

The weekly process of running forward DNS scan in Project Sonar is as follows:

1. **The list of domain names is compiled**
   Using the sources above and several others (which are not publicly disclosed by Rapid7), a fresh list of domains is compiled every week. This list might contain non-existing domains because of the passive nature of some sources.

2. **Mass DNS resolving starts**
   Rapid7 uses *pycares* [3] for mass DNS resolving. It is a Python interface to *c-ares*, which is a C library that performs DNS requests and name resolutions asynchronously. For every domain name in the list, DNS request for ANY meta-query type is made.

3. **Responses are saved**
   From the responses, a dataset is created, compressed using GZIP, and uploaded to scans.io repository. Note, however, that only domain names that responded with some data are included in the final dataset. In other words, domain names that responded with an error status, such as NXDOMAIN or SERVFAIL are not included in the final dataset.

---

3. https://pypi.python.org/pypi/pycares

One of the problems with this approach is that some DNS servers are not responding to ANY meta-query. *CloudFlare* is one of the DNS providers disabling ANY because of the increased load on their infrastructure [GM15]. Domains that do not respond to ANY requests are still included in the final dataset. It is at least an indication that a domain is active and can send a valid response when iterative resolving (requesting each resource record type separately) is used.

As stated before, Project Sonar is much more suitable for domain name analysis than port 53 scan from Censys. The dataset is uploaded periodically, which can be leveraged to analyze historical data. For the reasons listed above, Forward DNS dataset (referred to as **FDNS** from now on) from Project Sonar was chosen as a primary dataset, which is used in the current and the following chapters of this thesis.

3.3   Dataset analysis

FDNS is a GZIP compressed text file. Every line contains single DNS response in JSON format. Uncompressed, it looks as follows:

```
{"timestamp":"1506765777","name":"example.com","type":"a",
"value":"93.184.216.34"}

{"timestamp":"1506765777","name":"example.com","type":"aaaa",
"value":"2606:2800:220:1:248:1893:25c8:1946"}

{"timestamp":"1506788264","name":"example.com","type":"ns",
"value":"a.iana-servers.net"}

{"timestamp":"1506788264","name":"example.com","type":"ns",
"value":"b.iana-servers.net"}
```

As presented in the example above, domain names that have multiple resource records available have these records spread across multiple lines. For *example.com* domain, it means A, AAAA, and two NS records. Each line contains at least four key-value pairs:

- **Timestamp** – UNIX timestamp representing the time when the response was received

- **Name** – Domain name resolved (*Name* field of resource record)

- **Type** – Type of DNS resource record (*Type* field of resource record)

- **Value** – Response from DNS resolver (*Data* field of resource record)

For the domains that do not allow ANY resolution the following line is included:

```
{
    "5":"See draft-ietf-dnsop-refuse-any",
    "timestamp":"1506672702",
    "name":"100.43.157.155.static.krypt.com",
    "type":"hinfo",
    "value":"ANY obsoleted"
}
```

As can be seen, *100.43.157.155.static.krypt.com* does not allow ANY meta-query. The text included in the *value* field depends on the particular DNS provider. These domains also include special resource type called *hinfo*. They can be easily filtered using this unique value.

One might notice that a single record in FDNS is, in fact, a key-value pair (key being *name* field, value being *value* field) with additional information (timestamp and type). This structure makes FDNS very efficient in processing (as will be seen in the next chapters). The *name* field always contains a domain name. Depending on the context, the *value* field contains either another domain name, IPv4 address, IPv6 address, or another text information (e.g., SOA, TXT, which are not used in this thesis). The processing tools and scripts can, therefore, rely upon that *name* field will exclusively contain a domain name.

At the time of the writing (October 2017), the FDNS has a size of 23GB, compressed. Uncompressed, this equals to approximately 2.2 billion lines. To efficiently process this volume of data, fast processing tools are needed to read, filter, and process the compressed text files containing multiple JSON lines. For this purpose, two main command-line tools were chosen:

- **zcat** [4]
  Because FDNS is compressed using GZIP, the first step is to decompress it. *zcat* is a command-line tool that takes a compressed file as an input and writes the uncompressed data on standard output. There is no need to decompress the full file at once; zcat can do it on-the-fly.

- **jq** [5]
  jq is a command-line processing tool for JSON objects. It is capable of parsing, filtering, and processing JSON objects in an easy way. jq can print only selected subset of keys, evaluate if-then-else statements, transform the data, and much more. Another benefit of jq is that it allows processing files with multiple JSON objects - exactly how FDNS is structured. For instance, to only print *name* field from FDNS, the following jq command can be used:

---

4. https://linux.die.net/man/1/zcat
5. https://stedolan.github.io/jq/manual/

```
zcat 20171027-fdns.json.gz | jq '.name'
```

Indeed, large data analytics systems, such as *ElasticSearch* [6] or *Splunk* [7] are much more suitable for storing and analyzing FDNS. However, because of an enormous amount of data, the multinode setup of these systems would be required, with an extended optimization, which is beyond the scope of this thesis.

The rest of this chapter is devoted to basic statistics about FDNS. All statistics were calculated for FDNS which was collected October 27th, 2017.

Record types

The count of different values in the *type* field can be obtained using the following command:

```
zcat 20171027-fdns.json.gz
| jq -r '.type'
| sort
| uniq -c
| sort -nr
```

Type	Count
A	1 287 593 985
NS	423 240 252
MX	239 806 422
SOA	163 015 182
TXT	84 559 989
CNAME	31 783 410
AAAA	20 028 730
HINFO	17 934 073
PTR	2 765 763
UNK_IN_47	2 467 244

Table 3.1: Top 10 DNS record types in FDNS. As can be seen, compared to A records, a number of CNAME records is pretty low. Still, FDNS gives a good amount of CNAME records for practical analysis, as is discussed in *chapter 5*.

Table 3.1 shows the result for different field *type* values. Overall, there are 86 distinct values. A large subset of these values is taken by various error codes that *pycares* generated because it was unable to retrieve the answer (usually due to RRSIG records). These error

---

6. https://www.elastic.co/products/elasticsearch
7. https://www.splunk.com/

codes are labeled as *UNK_IN_* followed by a number. It is good to note that field *type* does not necessarily contain only DNS resource record types, but also error status codes.

As can be seen, FDNS contains almost 18 million *HINFO* records. These indicate that the domain name does not support ANY meta-query.

Another interesting observation is that FDNS contain PTR records. This is because *.in-addr.arpa* domain names are also included in the initial domain list. However, PTR records for such addresses point to the same domain, so the majority of PTR records have the same string in *name* and *value* fields. See example here:

```
{"timestamp":"1509091424","name":"0.99.221.166.in-addr.arpa",
"type":"ptr","value":"0.99.221.166.in-addr.arpa"}
```

Domain names

To obtain a list of all resolvable domain names in FDNS, the following command can be used:

```
zcat 20171027-fdns.json.gz
| jq -r '.name'
| sort
| uniq
```

The result yields into *1 225 115 767* domain names. This statistic, however, contains all domain names, not only base domains. Base domains can be extracted using the following command:

```
zcat 20171027-fdns.json.gz
| jq -r '.name'
| python base_domain.py
| sort
| uniq > base_domains.txt
```

The Python script is available in the attached archive (see Appendix A). It transforms the domain name to the base domain (e.g., *sub.sub.example.com* to *example.com*). This process cannot be done just by splitting the string using "." (dot). Some top-level domains have multiple levels, for instance, *.co.uk*. The Python script, therefore, contains a list of

public top-level domains [8], which is used to transform a domain name to a base domain correctly.

There are *182 658 426* different base domains in FDNS. From the base domains list, a distribution of top-level domains can be extracted using the command below.

```
cat base_domains.txt
| python tld_extract.py
| sort
| uniq -c
| sort -nr
```

The Python script (*tld_extract.py*) transforms a base domain to top-level domain (e.g., *example.com* to *com*) using the same technique as explained in the previous command. The distribution of top-level domains (for base domains) is listed in the Table 3.2.

TLD	Records
.COM	116 276 677
.NET	13 189 337
.ORG	9 560 209
.DE	7 318 847
.INFO	5 065 757
.RU	4 455 497
.US	1 918 097
.CO.UK	1 809 342
.CN	1 404 176
.BIZ	1 322 642

Table 3.2: Top 10 top-level domains in FDNS. This statistic was retrieved from unique base domains.

There is also a public project called *dnspop* [9], which makes statistics for most common prefixes of domain names in FDNS. These prefixes are heavily discussed in subsection 4.1.1.

CNAME records

Since *chapter 5* deals almost exclusively with CNAME records, it is useful to retrieve the list of the most popular CNAME records. To compute the usage of various base domains in CNAME records, the command below can be used.

---

8. https://publicsuffix.org/
9. https://github.com/bitquark/dnspop

```
cat base_domains.txt
| python tld_extract.py
| sort
| uniq -c
| sort -nr
```

Note, however, that only *value* field is taken into account. The reason is that the *value* field defines to where the CNAME record is delegated. Table 3.3 represents most used providers for CNAME records. As noted above, this statistic is utilized in chapter 5.

Base domain	Records
dcoin.co	1 976 289
dragonparking.com	1 373 884
wordpress.com	1 227 920
googleusercontent.com	1 014 623
amazonaws.com	973 440
jinmi.com	957 027
dnsdun.com	854 111
ename.net	485 628
google.com	323 231
weebly.com	236 455

Table 3.3: Top 10 base domains used in *value* field of CNAME records.

All commands used in this section are available in the attached archive as *data_analysis.sh* script.

4 Domain correlation

This chapter introduces ideas and techniques for correlating multiple domain names related to the single entity. It is a process of finding domain names, which are different but related to the same person or organization. For instance, *www.google.com*, *mail.google.com*, and *youtube.com* are different domain names. However, all three are associated with the same entity: *Alphabet Inc.*

Such correlation process takes place in a *reconnaissance phase* of the *kill chain* [Sag14]. The more domains the organization have publicly exposed, the larger attack surface is available for cyber adversaries. For instance, these domains might be vulnerable to subdomain takeover which is topic of *chapter 5*. Let's take an analogy with a house - the more the windows and doors, the higher the chance of housebreaking of the house. Even when the windows are tamperproof, there might be the case when one of them is opened for some period. The same thing applies in a cyber world.

The concepts presented in this chapter can help organizations understand their public exposure from the eyes of cyber adversaries. Such understanding further leads to an identification of weak spots and improvement of internal security processes.

Before dealing with technical details, definitions of vertical and horizontal domain correlation need to be established. Figure 4.1 depicts the difference.
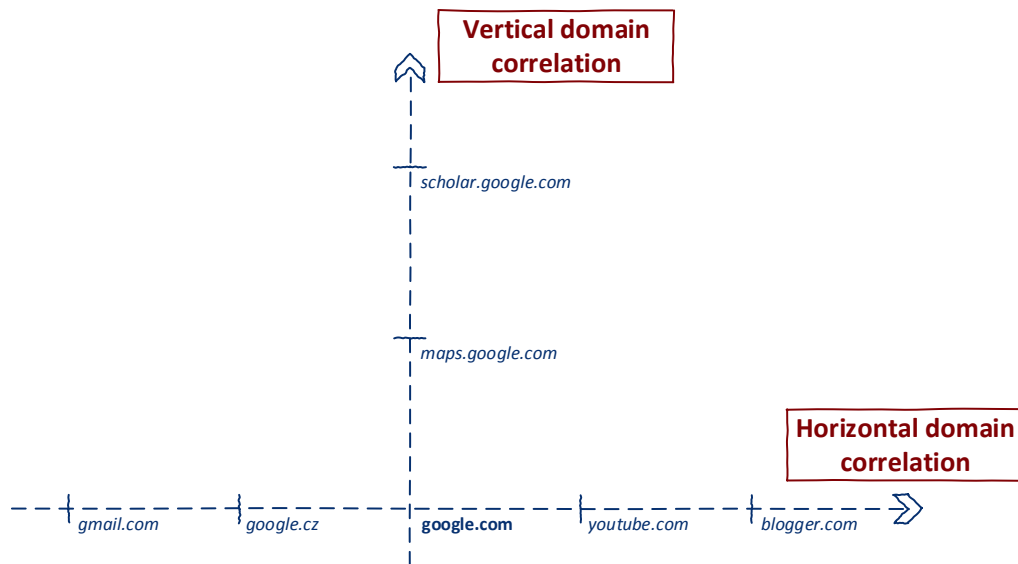

Figure 4.1: The difference between vertical and horizontal domain correlation for *google.com* domain.

- **Vertical domain correlation**
  Given the domain name, vertical domain correlation is a process of finding domains share the same base domain. This process is also called *subdomain enumeration.*

- **Horizontal domain correlation**
  Given the domain name, horizontal domain correlation is a process of finding other domain names, which have a different second-level domain name but are related to the same entity.

In the next sections, a word **target** denotes the entity of interest in the correlation process.

4.1 Vertical domain correlation

The vertical domain correlation process tries to find as many domain names specified in the zone file of the target as possible. The most basic approach is to dump the target's DNS zone file. Due to the sensitive information residing inside zone files, administrators in most cases configure DNS servers in a way that the zone files cannot be obtained by the regular hosts on the Internet [CR13]. AXFR transfer (zone transfer) is usually allowed only between primary and secondary DNS servers. Sometimes, however, AXFR transfer still works. AXFR transfer can be checked using a simple *dig* command:

```
dig axfr @dns.server domain.name
```

For most domains, this approach fails [Int15]. Therefore, different vertical correlation techniques need to be used. The techniques presented in the next sections are meant to demonstrate the different approaches which can be used for this process. There is not the only technique that will provide the best result set for every target.

4.1.1 Dictionaries

In password cracking terminology, a dictionary attack is seen as a limited brute-force attack [Yia13]. Vertical domain correlation can be in some sense seen as a refined password cracking problem. In vertical domain correlation, the goal is to find all domain names which are defined in target's DNS zone, while the content of the zone file is unknown. Figure 4.2 illustrates this analogy. DNS resolvers are used to verify whether some domain exists or not (in other words, whether it is present in DNS zone or not). The same binary decision happens during password cracking. Therefore, dictionary attacks can be used as one of the techniques in vertical domain correlation.

The process of using dictionary technique can be simplified into two steps:

1. **Creating the dictionary**
   A relevant dictionary (*wordlist*) of possible domain name prefixes needs to be created. Prefixes are the substrings of domain names without the base domain. For instance, one of the prefixes of *google.com* is *images*, because *images.google.com* is an existing domain (October 2017). One of the approaches for creating the wordlist
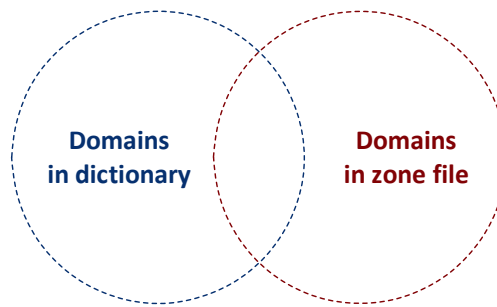
Figure 4.2: Illustration of domain correlation technique using sets. The main goal of vertical domain correlation using dictionaries is to find as large intersection between these two sets as possible.

is to take into account the popularity of domain prefixes. Fortunately, there exist several already created wordlists which are suitable for vertical domain correlation (refer to section 3.3). *Daniel Miessler's SecList* is one of them [Mie17]. A different technique for generating domain wordlist using Markov chains was presented in the paper by Wagner et al. [Wag+12].

2. **Evaluating the dictionary**
   The authoritative DNS server needs to be queried for every domain name from the wordlist to determine whether the domain is present in target's DNS zone or not. This resolution process can be done by using mass DNS resolution tool (e.g. *massdns* [1]) or by some specific tool for DNS resolution using dictionaries (e.g. *subbrute* [2] or *fierce* [3]). DNS resolvers also need to be taken into account. Mass DNS resolution tools often do round-robin between different DNS resolvers. The reason behind this is, that after some amount of requests in a small period the DNS resolver might stop responding. Another reason is censorship. Some DNS resolvers might be injecting false DNS responses to prevent access to the desired domain [Lev12]. Tools such as *massdns* by default contain a list of open (recursive) DNS resolvers. Another good source for open resolvers is the website *public-dns.info* [4].

4.1.2 SSL certificates

As presented in section 3.2, X.509 certificates contain extension called *SubjectAltName*. This extension allows specifying multiple domain names in a single certificate. It is pretty unusual for non-related domains to share one certificate. Therefore these domains will be almost always associated with the same entity. Getting a list of subdomains using SSL certificates starts with gathering all certificates issued to the target. This process can be done manually, by downloading the certificates on target's known domains and parsing out the data. Another approach is to use one of the SSL datasets from scans.io, either Censys or

---

1. https://github.com/blechschmidt/massdns
2. https://github.com/TheRook/subbrute
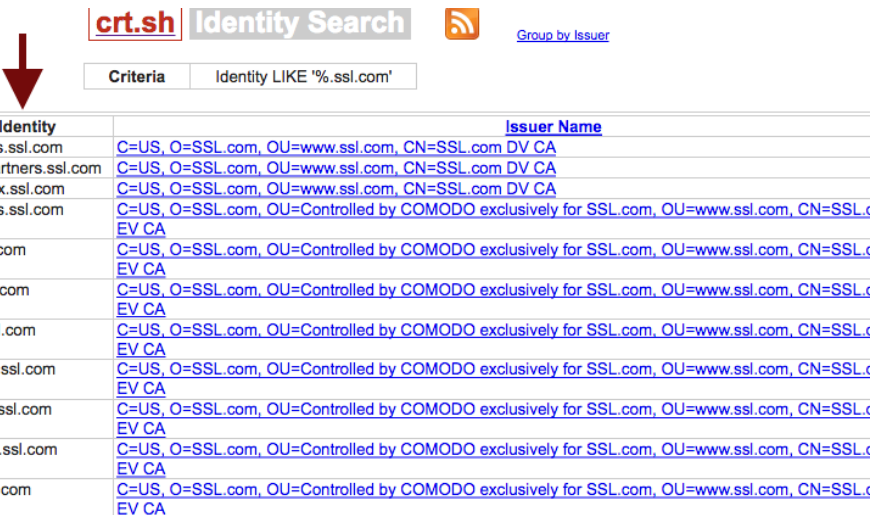3. https://github.com/mschwager/fierce
4. https://public-dns.info

Project Sonar. These datasets contain dumps of SSL certificates seen across scanned hosts. Unfortunately, as described in section 3.2, these certificates are collected by scanning IPv4 hosts (and not hostnames); hence many certificates might be missing in the dataset.

List of domain names associated with a certificate for *www.ssl.com* can be obtained with the following command (available in the attached archive as *san.sh*):

```
true
| openssl s_client -servername www.ssl.com -connect www.ssl.com:443
| openssl x509 -noout -text
| grep -Eo 'DNS:[^,]+'
| cut -c5-
```

This command returns all domain names specified in the *SubjectAltName* extension of SSL certificate. These domain names, however, are not limited to subdomains. The technique using SSL certificates can, therefore, be used both for vertical and horizontal domain correlation.

There is also a free web service called *crt.sh*, which gathers certificates submitted to *Certificate Transparency project* [LLK13]. Because crt.sh uses a different approach for collecting certificates, it should provide additional data compared to datasets from scans.io. The search functionality on crt.sh allows specifying the query using a wildcard. To find every certificate for some domain, *%.* prefix can be used to indicate all subdomains. Figure 4.3 illustrates this approach.

Figure 4.3: Search results for *%.ssl.com* on *crt.sh*. The screenshot shows different domain names found inside the certificates.

4.1.3 Search engines

Search engines are another excellent source of subdomain information. Since search engines are periodically crawling the target's websites, new subdomains are often present in HTML elements such as *anchor links*. In Project Sonar, one of the sources for FDNS is data from HTML. However, this is a limited set of data compared to what search engines have available. Search engines can crawl the website several levels deep, which might reveal new subdomains not seen on the index pages. The process of searching for these subdomains is pretty straightforward. Google allows specifying advanced operators in their search queries [LW07]. Another name for this technique is *Google dorking.* By using the query *"site:ssl.com"*, results from Google will be limited only to pages on *ssl.com* and all of its subdomains [Goo17b]. From there, it is easy to extract unique subdomains found by Google. This technique can also be used in *Bing* and *Yahoo.*
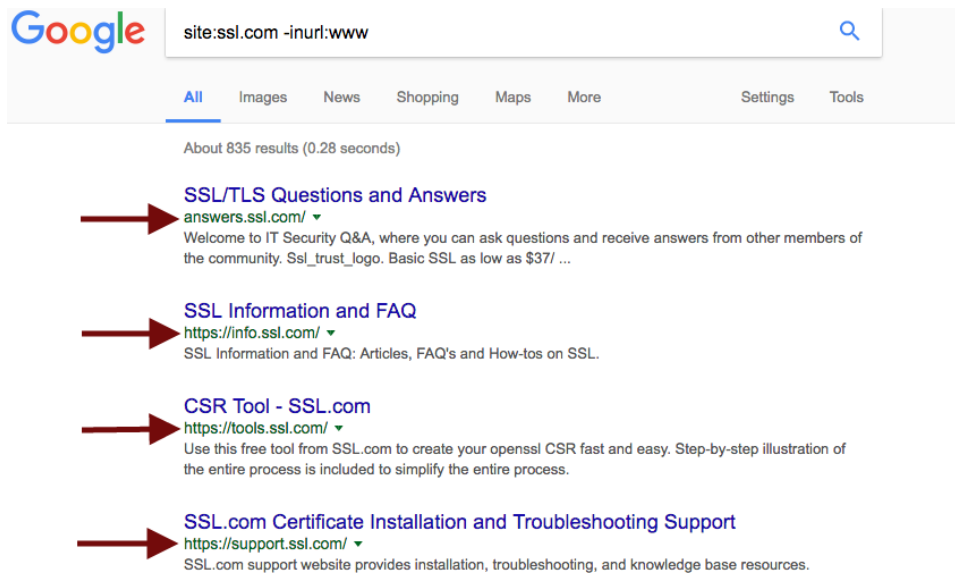

Figure 4.4: Google result page reveals different subdomains for *ssl.com*

Similar to Internet-wide scans (presented in section 3.1), some projects are periodically crawling the web and freely publishing the results. One of those projects is *Common Crawl.* Common Crawl is a non-profit organization which runs several crawls of the surface web every year. This might be seen as an alternative to the regular search engines such as Google, Bing, or Yahoo. However, Common Crawl makes the crawled data available to the public. The crawl from September 2017 contained 3.01 billion web pages and over 250 TB of uncompressed content [Nag17].

For vertical domain correlation process, Common Crawl data can be used to extract subdomains. Common Crawl provides public REST API for querying different information. One of the possible queries is an extraction of URLs that were visited on some particular domain. Luckily, this domain can be specified using a wildcard, which means that results that are obtained from Common Crawl will include different subdomains. Common Crawl REST API follows the specification of *CDX Server* API [Kre17]. The output of this REST

API are all resources that were crawled for the specified domain. Example JSON describing one resource looks like this:

```json
{
    "urlkey": "com,ssl)/",
    "timestamp": "20170922005637",
    "filename": "...20170922023402-00327.warc.gz",
    "mime-detected": "text/html",
    "status": "200",
    "mime": "text/html",
    "digest": "TDIYLMDHFWCTJ7HQOSSW2FT5O7RE7ZXV",
    "length": "13251",
    "offset": "858583875",
    "url": "https://www.ssl.com/"
}
```

For obtaining a list of subdomains for *ssl.com*, the following shell command can be used (available in the attached archive as *common_crawl.sh*):

```bash
http -b GET 'INDEX_URL/coll-cdx?url=*.ssl.com&output=json'
| jq -crM '.url'
| awk -F/ '{print $3}'
| awk -F\? '{print $1}'
| sort
| uniq
```

INDEX_URL needs to be replaced with an API endpoint of the chosen crawl. API endpoints can be found on *Common Crawl Index Server* [5] website. The command above will first download all resources (specified in JSON format shown above) that were crawled by Common Crawl (using *httpie* [6] and *jq*). Next, it extracts unique domain names out of all URLs. Example uses *awk*, *sort*, and *uniq* to do so.

4.1.4 Sublist3r

Open source tools dealing with vertical domain correlation process combine several techniques to improve the result set. One of the leading open source tools for this task is *Sublist3r* [7]. Sublist3r, written in Python, uses more than ten sources to obtain the subdomains. These sources include dictionaries (includes *subbrute* within), threat intelligence
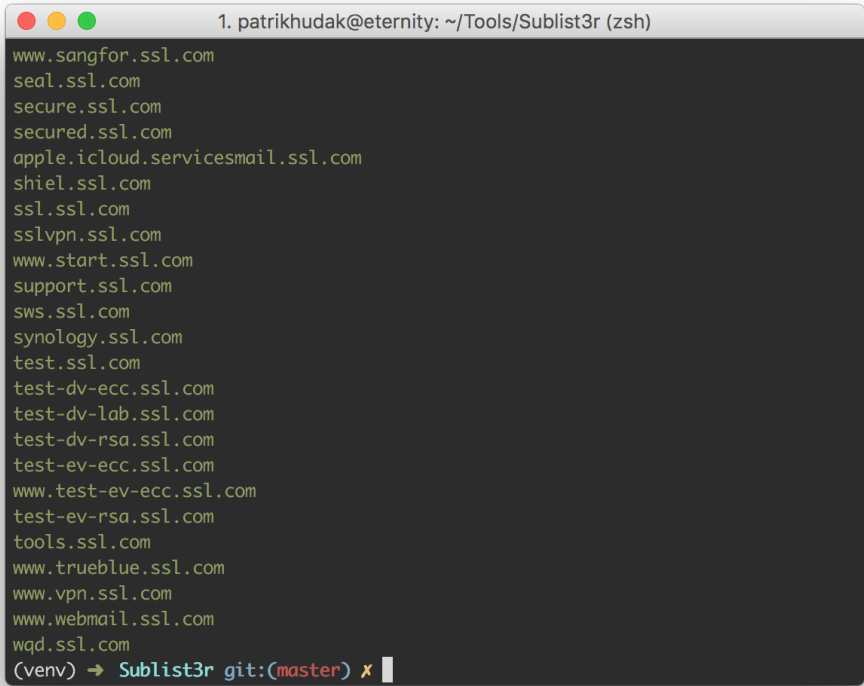
---

5. http://index.commoncrawl.org/
6. https://httpie.org/
7. https://github.com/aboul3la/Sublist3r

26

platforms (*Virustotal, ThreatCrowd*), and search engines (*Google, Yahoo, Bing*). The full list of sources along with details can be found on project's homepage. To get subdomains of *ssl.com* using Sublist3r, the following list of commands can be used:

```
git clone https://github.com/aboul3la/Sublist3r.git
cd Sublist3r
virtualenv venv
source venv/bin/activate
pip install -r requirements.txt
python sublist3r.py -d ssl.com
```

Commands above will download and install Sublist3r. The last command starts vertical domain correlation for *ssl.com*. The process might take a while, depending on the scope of the domain. In the end, Sublist3r combines all results to one list (as shown in on Figure 4.5). Note however that Sublist3r is also using passive sources. Therefore, the result set might contain domains that are no longer active (respond with NXDOMAIN status code). Tools such as *massdns* are used to exclude the inactive domains.

Figure 4.5: Partial results from Sublist3r for *ssl.com* domain

4.1.5 FDNS for vertical domain correlation

One of the problems with Sublist3r is that it cannot be used in scale. It uses sources such as Google which will block the usage of Sublist3r after a few tries because of strange network traffic. In section 3.2, a list of sources that Rapid7 uses to build its list for DNS resolution is presented. This list is quite similar to sources of Sublist3r: HTML elements, passive DNS data, SSL certificates and so on.

FDNS can also be used for vertical domain correlation. It is available offline; hence it can be used multiple times without any potential blockage. For finding all subdomains in FDNS dataset, the following command can be used (available in the attached archive as *fdns_vertical.sh*):

```
zcat 20170929-fdns.json.gz
| grep -F '.ssl.com"' # double quotes indicate end-of-word
| jq -crM 'if (.name | test("\\.ssl\\.com$")) then .name else empty end'
| sort
| uniq
```

In the command above, there are two filters used: *grep* and *jq*. The reason behind this is speed. While *jq* needs to parse every single line as JSON, *grep* is used to efficiently find lines with string *.ssl.com* within. Therefore, only lines where *.ssl.com* is found as a substring are forwarded to *jq*. This approach is much more efficient than parsing every single line as JSON. Afterwards, *jq* is testing the correct form of a domain by running regular expression which translates to "*everything that ends with .ssl.com*". In the end, the result set is sorted by *sort* and only unique values are kept (using *uniq*).

Domain	Sublist3r	FDNS
google.com	749	15 069
muni.cz	1 145	33 174
ieee.org	581	1 299
ssl.com	271	40
python.org	58	40

Table 4.1: A count of subdomains found using vertical domain correlation. The number includes only subdomains which are still active (October 2017).

As seen on Table 4.1, Sublist3r does not necessarily find more domains than FDNS. Large domains tend to have more results in FDNS while small domains tend to have more results provided by Sublist3r. For vertical domain correlation purposes, Sublist3r and FDNS should be used hand-by-hand to enhance each other's capabilities. For some domains, FDNS yields more results because of specific sources that Rapid7 is using for domain name list. For others, Sublist3r is able to find more because of its own specific sources.

4.2 Horizontal domain correlation

Horizontal domain correlation process tries to find all domain names which have different domain name structure but are associated with the target. As opposed to vertical domain correlation, the horizontal domain correlation cannot rely on finding substrings of some domain names or use dictionary attacks (the set of possible options is too big). Therefore, it is not a trivial problem to solve.

It is important to note, that techniques presented in this section might work only for large domain names. Another problem is, that even if some results set is obtained, it might contain false positives, i.e., domain names which are not associated with the target. In vertical correlation, when the subdomain is found and has the base domain equal to one of the target's domains, it is associated with it because of how DNS delegation works. However, this is not the case in horizontal domain correlation.

4.2.1 Naive approach

One of the simplest ideas is to rely on virtual hosting setup. To postpone IPv4 address exhaustion, virtual hosting method was created. It is an idea to host multiple domain names on a single host (having a single IPv4 address) [The17]. The single host might be hosting two different domains (such as *google.com* and *youtube.com* having the same IP address in A record). Because the two domains are hosted on the same host, common sense might say that they are associated with the same entity.

This is not necessarily true because of widespread usage of web hostings and cloud providers. One host might be serving tens or hundreds of domains for different entities. The IP address of a host is owned by hosting or cloud provider while domains are registered separately by regular organizations. Figure 4.6 illustrates this situation. Hence, the naive approach cannot be used reliably to assume that domains which are hosted on the same host are also related to each other.
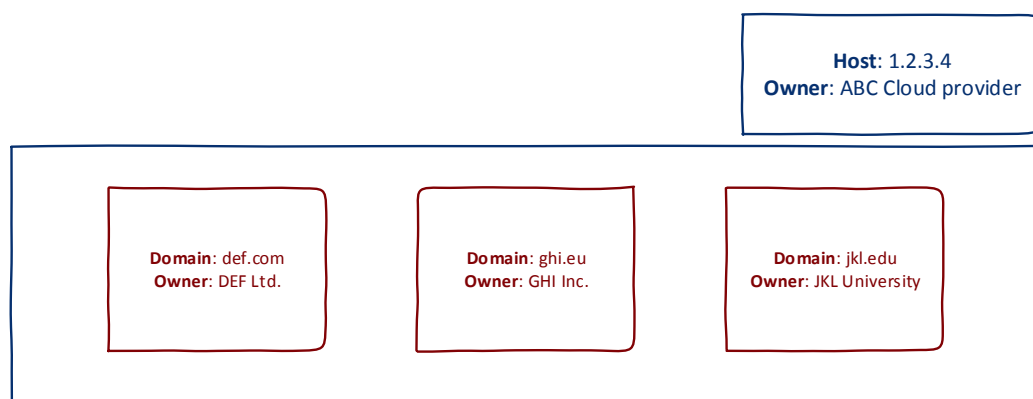

Figure 4.6: Virtual hosting setup on the public cloud provider. Even though three domains share one host, they are not associated with the same entity by any means.

4.2.2 Dedicated IP range

To prevent the false positives from the situation described in the section above, the only way the naive approach can work is when the IP address is also owned by the target. The organization called *IANA* allocates IP addresses to *Regional Internet Registries (RIRs)* [Küh15]. Organizations can request portion of IPv4 space from one of the RIRs. This is a common practice for larger organizations and academic institutions. For instance, one of Google's autonomous system is *AS15169* (see Figure 4.7), registered by *ARIN* (RIR in North America). All domains which have A records pointing to one of IP addresses in this autonomous system should be thus owned by the same entity, in this case, Google / Alphabet Inc. Note that however, any domain can set its A record to point to this IP range. One must take into account cases where the company is also a cloud provider. This is the situation for Google, with their *Google Cloud Platform*. Luckily, Google has several autonomous systems, each having its purpose. The process of finding domain names based on their IP address range is described in subsection 4.2.4.
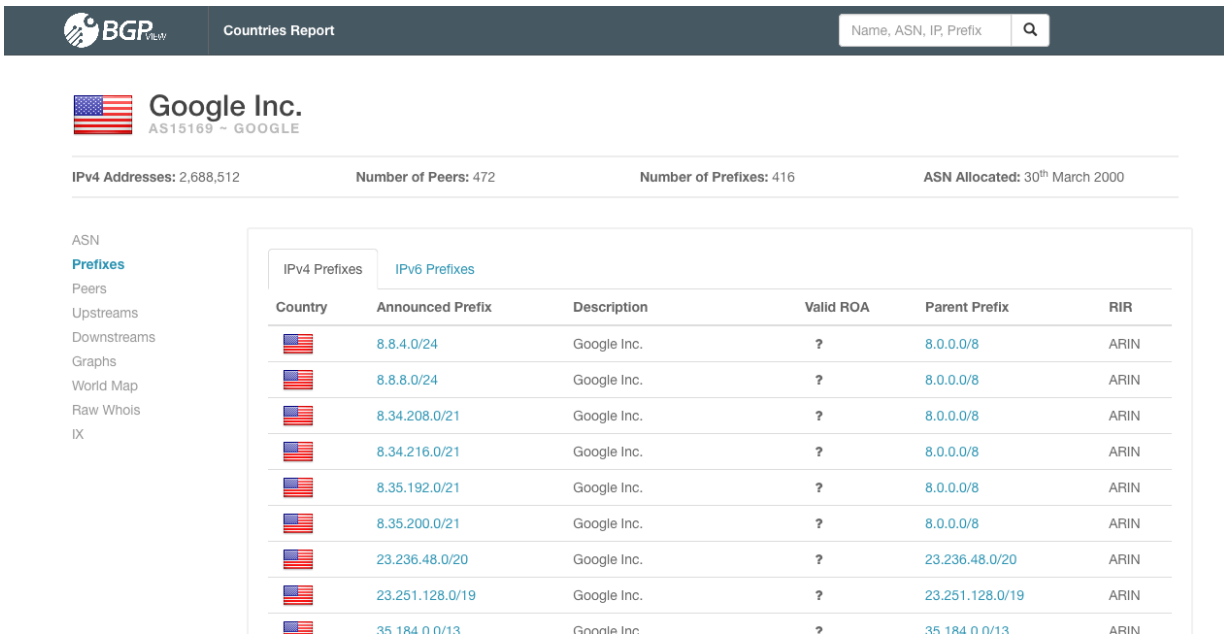
Figure 4.7: Listing of IPv4 prefixes for *AS15169*, owned by Google.

4.2.3 Reverse WHOIS

The entirely different approach for doing horizontal domain correlation is to leverage WHOIS database [GW95]. When the domain name registered, the contact details of the registrant are provided to the particular domain registry. This usually includes company name, telephone number, and e-mail. For most TLDs, this information can be queried using WHOIS protocol. However, there are cases where this information is hidden [Blo08]. Reverse WHOIS technique tries to find all domains with some shared WHOIS information. If two domains share the registrant company name, there is a high chance that they are registered by the same company. Fang et al. used reverse WHOIS technique to proactively

find phishing domains associated with one entity [Fan+15]. The process of finding similar domains using reverse WHOIS looks as follows:

1. **Selecting common (pivot) field**
   The process starts with one domain which is often the primary domain of the target. WHOIS data is queried for the selected domain. One needs to find a good candidate (*pivot*) for the field that the whole search will depend on. One of the best candidates for the pivot is either organization's address or e-mail address. Figure 4.8 illustrates the selection for *muni.cz* domain.

```
contact:        SB:MUNI-UVT
org:            Masarykova univerzita, Ustav vypocetni techniky
name:           Masarykova univerzita, Ustav vypocetni techniky
address:        Botanicka 68a
address:        Brno
address:        60200
address:        CZ
e-mail:         muni-adm@ics.muni.cz      ⬅
registrar:      REG-GENREG
created:        10.08.2001 22:13:00
changed:        24.09.2015 08:51:04
```

Figure 4.8: Selecting e-mail address from WHOIS data for *muni.cz* domain.

2. **Choosing a reverse WHOIS provider**
   There are not many sites providing reverse WHOIS functionality. *DomainTools* provides various services around domain names, one of them is reverse WHOIS [8]. However, this is a paid service; the price will depend on how many domains it found. One of the free services providing reverse WHOIS results is *viewdns.info* [9].

3. **Obtaining results**
   As stated several times before, the final result set might contain false positives, i.e., domains which are not associated with the target but somehow got into reverse WHOIS results. The process of verifying false positives is from the most parts manual. This means verifying WHOIS records and website content of each domain. Figure 4.9 shows the results for *muni.cz* domain.

Although results from reverse WHOIS technique still might contain false positives, it often produces much better results than the dedicated IP space technique. Cloud providers are widely utilized, which means that even large companies share IP range from the pool of IP addresses owned by cloud providers. However, domain names are registered independently of the underlying infrastructure. This fact makes reverse WHOIS suitable for horizontal domain correlation.

---

8. https://reversewhois.domaintools.com/
9. http://viewdns.info/reversewhois/

Figure 4.9: Results from reverse WHOIS search on *viewdns.info*. E-mail address from WHOIS data was chosen as a common factor for *muni.cz* domain.

4.2.4 FDNS for horizontal domain correlation

This section explains an novel approach for doing horizontal domain correlation using FDNS. This approach was developed specifically for this thesis.

Unfortunately, FDNS does not contain WHOIS data. Therefore, a different approach needs to be used. To prevent as many false positives as possible, the following process is proposed:

1. Start with the initial domain (*pivot*). Find NS records for all of the pivot's subdomains.

2. Filter only those NS records that are subdomains of the pivot.

3. Search for every domain in FDNS which points to one of these nameservers using NS record.

One might think that clusters of domains which have common nameserver are enough to tie domains in the cluster to the same entity. However, there are multiple nameservers which are managing zones for several thousand unrelated domains. The process described above should prevent false positives for the following reason: If the base domain of target and its nameserver is the same, the target is likely to have own DNS servers in place (and not use managed DNS provider). In other words, these DNS servers are likely to host only domains which are owned by the target. This process is likely to work only for more prominent organizations or networks, which are capable of hosting their DNS infrastructure. For the smaller targets or targets that do not have own DNS infrastructure and dedicated IP space, using the reverse WHOIS process is a better choice.

To obtain the list of NS record for all of the pivot's subdomains, the following command can be used (assuming the pivot is *muni.cz*):

```
zcat 20171013-fdns.json.gz
| grep -F '.muni.cz"'
| jq -crM 'if (.name | test("\\.muni\\.cz$")) then . else empty end'
| jq -crM 'if .type == "ns" then .value else empty end'
| sort
| uniq
| grep -E '\.muni\.cz$' > muni_nameservers.txt
```

This yields to list of nameservers which are subdomains of the pivot (combining step one and two from the process described above). The last step is to reverse the search and look for all domain names which are pointing to one of these nameservers:

```
zcat 20171013-fdns.json.gz
| grep -F '.muni.cz"'
| jq -crM 'if .type == "ns" then . else empty end'
| python ns_group.py muni_nameservers.txt
| python base_domain.py
| sort
| uniq
```

The first grep is used as a basic filter. The matching record will have nameserver which is a subdomain of *muni.cz*. The following jq command filters only NS records. The core functionality lies in *ns_group.py* script (available in the attached archive). This Python script takes NS record and checks whether its *value* (in NS record context, this means nameserver) matches one of the nameservers obtained from the previous step. The result will include a list of all domains, not only base domains. Since this section deals exclusively with horizontal domain correlation, the subdomains will be ignored. The list of unique base domains can be obtained using a script called *base_domain.py* (available in the attached archive). The final list is then sorted, and only unique values are kept.

FDNS can also be used for finding domains pointing to IP addresses in some specific IP range (technique presented in subsection 4.2.2). Firstly, the specific IP range of the target needs to be chosen. Afterwards, the command-line tool called *grepcidr* [10] filters only those A records that have *value* field set to the IP address from the specified set. Instead of filtering IP addresses one by one, *grepcidr* helps with finding IP addresses specified by *Classless Inter-Domain Routing (CIDR)* notation.

---

10. https://github.com/frohoff/grepcidr

Masaryk University is part of *AS2852* and has IP range of 147.251.0.0/16 [11]. To find all domains pointing to this IP range, the following command can be used (available in the attached archive as *ip_range.sh*):

```
zcat 20171013-fdns.json.gz
| grepcidr '147.251.0.0/16'
| jq -crM 'if .type == "a" then .name else empty end'
| python base_domain.py
| sort
| uniq
```

Note that however, any domain can set its A to this range and thus introduce potential false positives in the output. Similarly as was explained in subsection 4.2.3, the result list should be manually verified to confirm that each domain is related to the target.

Domain	Reverse WHOIS	NS group	IP range
google.com	14121 (*dns-admin@google.com*)	3206	2543 (*172.217.0.0/16*)
muni.cz	20 (*muni-adm@ics.muni.cz*)	120	186 (*147.251.0.0/16*)
ieee.org	878 (*hostmaster@ieee.org*)	172	175 (*140.98.0.0/16*)

Table 4.2: Related domains found using different horizontal domain correlation techniques. The value in parentheses represents an argument which was used for a particular technique.

Results in Table 4.2 indicate that for some domains, one technique provides more results than the other. As was stated several times in this chapter, there is no best approach for every domain. Results need to be properly examined to see which technique yield to less false positives.

Usually, the process of finding all domain names related to the target uses horizontal and vertical domain correlation techniques iteratively. When the new domain name is found using horizontal domain correlation, the vertical domain correlation is run afterward to find all of its subdomains.

---

11. https://bgpview.io/prefix/147.251.0.0/16

5 Subdomain Takeovers

One of the extensive researches done around the concept of *subdomain takeovers* was done by Liu et al. in 2016 [LHW16]. This chapter starts with an explanation of subdomain takeover fundamentals and extends the previously mentioned research by utilizing FDNS.

Subdomain takeover is a process of registering a non-existing domain name to gain control over another domain. The most common scenario of this process follows:

1. Domain name (e.g. *sub.example.com*) uses a CNAME record to another domain (e.g. *sub.example.com CNAME anotherdomain.com*).

2. In some point in time, *anotherdomain.com* expires and is available for registration by anyone.

3. Since the CNAME record is not deleted from *example.com* DNS zone, anyone who registers *anotherdomain.com* has full control over *sub.example.com* until the DNS record is present.

The implications of the subdomain takeover can be pretty significant. Using subdomain takeover, attackers can send phishing emails from the legitimate domain, perform cross-site scripting (XSS), or damage the reputation of the brand which is associated with the domain. The implications are heavily discussed in section 5.4.

Such scenario is not purely hypothetical and section 5.3, in fact, shows the prevalence of subdomain takeover on the Internet. Depending on the context, subdomain takeover can be interpreted as a vulnerability, misconfiguration, or human error.

Subdomain takeover is not limited to CNAME records. NS, MX and even A records (which are not subject to this chapter) are affected as well. This chapter deals primarily with CNAME records. However, use cases for NS and MX records are presented where needed. Figure 5.1 explains the notation that will be followed throughout the rest of this chapter.
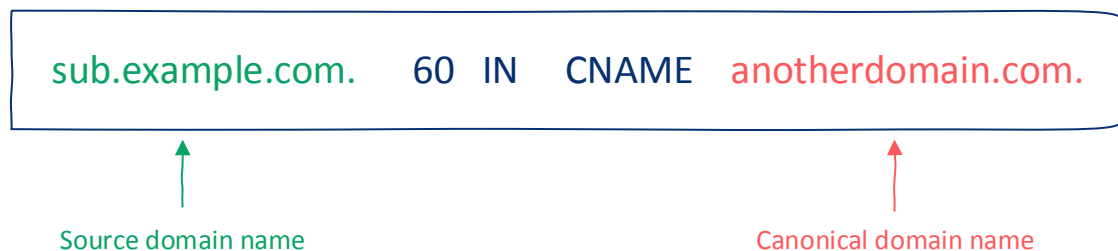

Figure 5.1: Notation used to refer to different parts of CNAME record.

5.1 Regular domains

DNS delegation using CNAME record is entirely transparent to the user, i.e., it happens in the background during DNS resolution. Figure 5.2 illustrates the behavior of web browser for the domain name which has CNAME record in place. Note that a web browser is implicitly putting trust to anything that DNS resolver returns. Such trust means that when an attacker gains control over DNS records, all web browser security measurements (e.g., *same-origin policy*) are bypassed [Ker16]. This presents a considerable security threat since subdomain takeover breaks authenticity of a domain which can be leveraged by an attacker in several ways. As will be shown in section 5.4, TLS/SSL does not fix this problem since subdomain takeover is not regular *Man-in-the-middle* style attack.
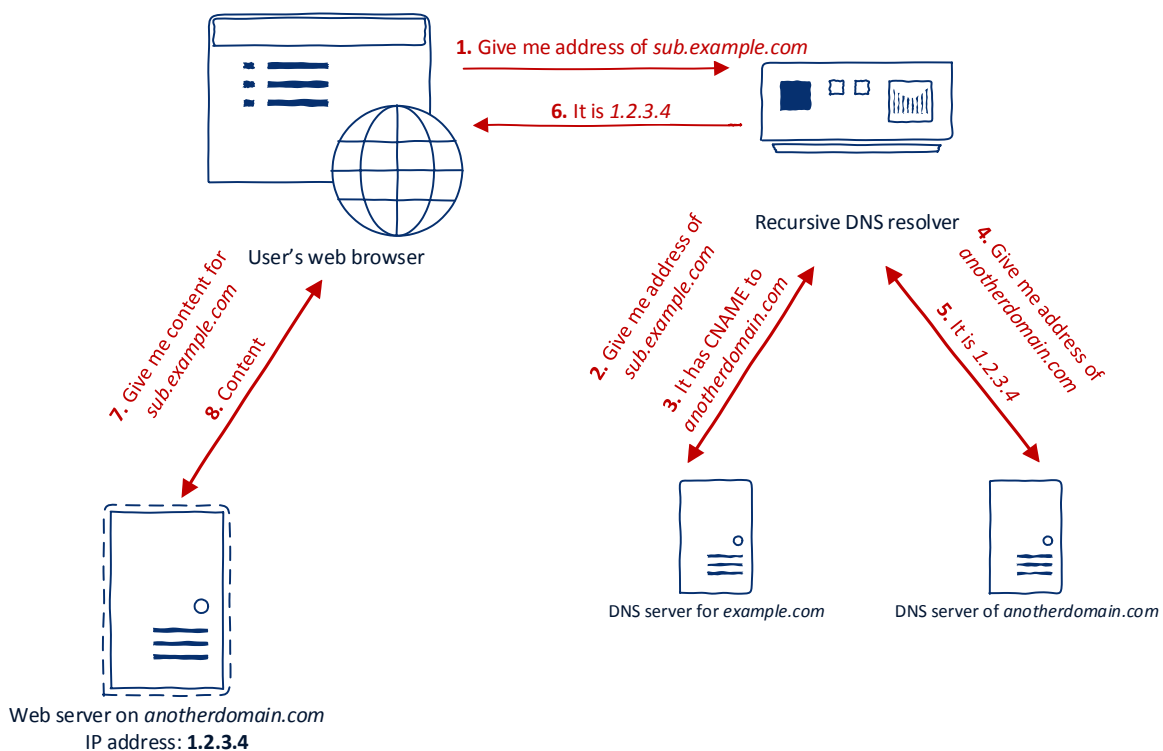
Figure 5.2: DNS resolution process from the perspective of a web browser. Note that step 7 requests *sub.example.com* rather than *anotherdomain.com*. That is because the web browser is not aware that *anotherdomain.com* even exist. Even though CNAME record is used, the URL bar in the browser will still contain *sub.example.com*.

CNAME subdomain takeover. One of the primary types of CNAME subdomain takeover is the scenario when a canonical domain name is a regular Internet domain (not one owned by cloud providers as will be explained in section 5.2).

The process of detecting whether some source domain name is vulnerable to CNAME subdomain takeover is quite straightforward:

Given the pair of source and canonical domain names, if the base domain of a canonical domain name is available for registration, the source domain name is vulnerable to subdomain takeover.
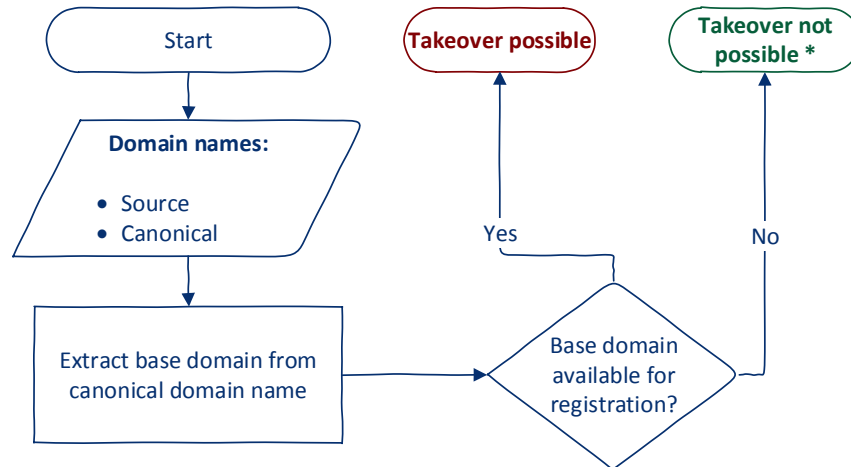
Figure 5.3: Flowchart describes a simple decision process to determine whether the source domain name is vulnerable to subdomain takeover.

Noteworthy thing in the process is „*the base domain of a canonical domain name"*. That is because the canonical domain name might be in the form of a higher-level domain. If a base domain is available for registration, the higher-level domain names can be easily recreated in the DNS zone afterward.

Checking the availability of base domain names can be achieved using domain registrars such as GoDaddy [1] or Namecheap [2]. One might think that testing a DNS response status for NXDOMAIN is sufficient indication that the domain name is available for registration. Note however that it is not the case since there are cases where domain name responds with NXDOMAIN but cannot be registered. Reasons include restricted top-level domains (e.g., .GOV, .MIL) or reserved domain names by TLD registrars.

NS subdomain takeover. The concept of subdomain takeover can be naturally extended to NS records: *If the base domain of canonical domain name of at least one NS record is available for registration, the source domain name is vulnerable to subdomain takeover.*

One of the problems in subdomain takeover using NS record is that the source domain name usually has multiple NS records. Multiple NS records are used for redun-

---

1. https://godaddy.com/
2. https://www.namecheap.com/

dancy and load balancing. The nameserver is chosen randomly before DNS resolution. Suppose that the domain *sub.example.com* has two NS records: *ns.vulnerable.com* and *ns.nonvulnerable.com*. If an attacker takes over the *ns.vulnerable.com*, the situation from perspective of the user who queries *sub.example.com* looks as follows:

- Since there are two nameservers, one is randomly chosen. This means the probability of querying nameserver controlled by an attacker is 50%.

- If user's DNS resolver chooses *ns.nonvulnerable.com* (legitimate nameserver), the correct result is returned and likely being cached somewhere between 6 and 24 hours.

- If user's DNS resolver chooses *ns.vulnerable.com* (nameserver owned by an attacker), an attacker might provide a false result which will also be cached. Since an attacker is in control of nameserver, she can set TTL for this particular result to be for example one week.

The process above is repeated every time the cache entry expires. When an attacker chooses to use TTL with high value, the false result will stay in DNS cache for that period. During this time, all requests to *sub.example.com* will use false DNS result cached by an attacker. This idea is even amplified when public DNS resolvers (e.g., Google DNS) are used. In this case, public resolvers are likely to cache the false results which means that all users using the same DNS resolver will obtain false results until the cache is revoked.

In addition to control over the source domain name, control over all higher-level domains of source domain name is gained as well. That is because owning a canonical domain name of NS record means owning the full DNS zone of the source domain name.

In 2016, Matthew Bryant demonstrated a subdomain takeover using NS record on *maris.int* [Bry16]. The .INT top-level domain is a special TLD, and the only handful of domains are using it. Bryant showed that even though registration of such domain names is approved exclusively by IANA, nameservers can be set to arbitrary domains. Since one of *maris.int* nameservers was available for registration (*cobalt.aliis.be*), subdomain takeover was possible even on this restricted TLD.

Bryant also demonstrated even higher severity attack where he was able to gain control over nameserver of .IO top-level domain [Bry17]. Gaining control over .IO means controlling responses for all .IO domain names. In this case, one of .IO nameservers was *ns-a1.io* which was available for registration. By registering *ns-a1.io* Bryant was able to receive DNS queries and control their responses for all .IO domains.

MX subdomain takeover. Compared to NS and CNAME subdomain takeovers, MX subdomain takeover has the lowest impact. Since MX records are used only to receive e-mails, gaining control over canonical domain name in MX record only allows an attacker to receive e-mails addressed to source domain name. Although the impact is not as big as for CNAME or NS subdomain takeover, MX subdomain takeover might play role in spear phishing attacks (as explained in section 5.4) and intellectual property stealing.

5.2 Cloud providers

Cloud services are gaining popularity in recent years [Wei17]. One of the basic premises of the cloud is to offload its users from setting up their own infrastructure. Organizations are switching from on-premise setup to alternatives such as cloud storage, e-commerce in a cloud, and platform-as-a-service, just to name a few.

After a user creates a new cloud service, the cloud provider in most cases generates unique domain name which is used to access the created resource. Because registering a domain name via TLD registrar is not very convenient because of a large amount of cloud service customers, cloud providers opt to use subdomains. The subdomain identifying unique cloud resource often comes in the format of *name-of-customer.cloudprovider.com*, where *cloudprovider.com* is a base domain owned by the particular cloud provider.

If the cloud service registered by an organization is meant to be public (e.g., e-commerce store), the particular organization might want to have it present as part of their domain. The main reason behind this is branding: *shop.organization.com* looks better than *organization.ecommerceprovider.com*. In this case, the organization has two choices:

- **HTTP 301/302 redirect**
  301 and *302* are HTTP response codes that trigger a web browser to redirect the current URL to another URL. In the context of cloud services, the first request is made to a domain name of an organization (e.g., *shop.organization.com*) and then redirect is made to a domain name of cloud providers (e.g., *organization.ecommerceprovider.com*).

- **CNAME record**
  Using this method, the „redirect" happens during DNS resolution (recall Figure 5.2). The organization sets CNAME record and all traffic is automatically delegated to the cloud provider. Using this method, the URL in the user's browser stays the same. Note however that the particular cloud service must support delegation using CNAME records.

If the CNAME record method is used, the possibility of subdomain takeovers comes into play. Even though the cloud provider owns the base domain of a canonical domain name, subdomain takeover is still possible as is presented in the next sections.

The providers in the subsequent sections were chosen based on three primary reasons:

1. **Prevalence**
   Based on section 3.3 statistics on CNAME records, cloud providers domains with the highest usage in CNAME records were prioritized.

2. **Support for CNAME records**
   As explained above, cloud provider needs to support CNAME delegation. Cloud providers realize that customers request such behavior and the most popular cloud providers already support it.

3. **Domain ownership verification**
   The chosen cloud providers are not verifying the ownership of the source domain

name. Since the owner does not need to be proven, anyone can use expired cloud configuration to realize subdomain takeover.

5.2.1 Amazon CloudFront

Amazon CloudFront is a *Content Delivery Network (CDN)* in Amazon Web Services (AWS) [Ama17b]. CDN distributes copies of web content to servers located in different geographic locations (called points of presence). When a user makes a request to CDN, the closest point of presence is chosen based on visitors location to lower the latency [Imp17]. CDNs are utilized by organizations, mainly to distribute media files such as video, audio, and images. Other advantages of CDNs include Denial of Service attacks protection, reduced bandwidth, and load balancing in case of high traffic spikes.

CloudFront uses *Amazon S3* [3] as a primary source of web content [Ama17b]. Amazon S3 is another service offered by AWS. It is a cloud storage service (S3 is an abbreviation for Simple Storage Service) which allows users to upload files into so-called *buckets*, which is a name for logical groups within S3.

CloudFront works with the notion of *distributions*. Each distribution is a link to specific Amazon S3 bucket to serve the objects (files) from. When the new CloudFront distribution is created, a unique subdomain is generated to provide access [Ama17c]. The format of this subdomain is *SUBDOMAIN.cloudfront.net*. The *SUBDOMAIN* part is generated by CloudFront and cannot be specified by a user.

In addition to a randomly generated subdomain, CloudFront includes a possibility to specify an alternate domain name for accessing the distribution [Ama17c]. This works by creating CNAME record from alternate domain name to subdomain generated by Cloud-Front. Although Amazon does not provide documentation about the internal CloudFront concepts, the high-level architecture can be deducted from its behavior. Based on the geographic location, DNS query to any subdomain of *cloudfront.net* leads to the same A records (in the same region). This indicates that CloudFront is using the virtual hosting setup in the backend. After the HTTP request arrives, CloudFront's edge server determines the correct distribution based on HTTP *Host* header. Documentation also supports this theory as it states: *„You cannot add an alternate domain name to a CloudFront distribution if the alternate domain name already exists in another CloudFront distribution, even if your AWS account owns the other distribution""* [Ama17c]. Having multiple alternate domains pointing to one distribution is correct, however, having the same alternate domain name present in multiple distributions is not. Figure 5.4 illustrates this concept.

Therefore to correctly handle alternate domain names, CloudFront needs to know beforehand to which distribution the alternate domain name is attached. In other words, having CNAME record configured is not enough, the alternate domain name needs to be explicitly set in distribution settings.

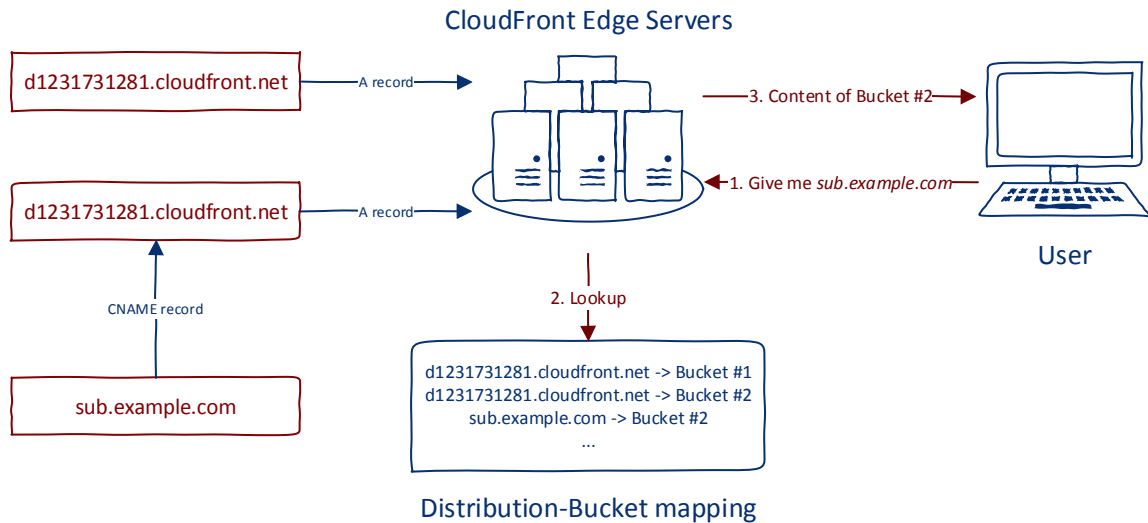3. https://docs.aws.amazon.com/AmazonS3/latest/dev/Welcome.html

Figure 5.4: High-level architecture of CloudFront. After user makes HTTP request, mapping table is used to correctly determine the S3 bucket.

The problem with alternate domain names in CloudFront is similar to problems explained in section 5.1. Let's assume that *sub.example.com* has a CNAME record set to *d1231731281.cloudfront.net*. When there is no *sub.example.com* registered in any CloudFront distribution as an alternate domain name, subdomain takeover is possible. Anyone is able to create a new distribution and set *sub.example.com* as an alternate domain name in its settings. Note that however, the newly created CloudFront subdomain does not need to match the one specified in the CNAME record (*d1231731281.cloudfront.net*). Since CloudFront uses a virtual hosting setup, the correct distribution is determined using HTTP Host header and not DNS record.

Figure C.2 shows the error message that is presented after HTTP request to alternate domain name which has the DNS CNAME record to CloudFront in place but is not registered in any CloudFront distribution. This error message is very strong indication of possibility of subdomain takeover. Nevertheless, the two exceptions need to be taken into account:

- **HTTP / HTTPS only distributions**
  CloudFront allows specifying whether the distribution is HTTP-only or HTTPS-only. Switching HTTP to HTTPS might provide correct responses for some distributions.

- **Disabled distribution**
  Some distributions might be disabled. Disabled distribution is no longer actively serving content while still preserving its settings. It means that some alternate domain name might be throwing an error message after HTTP request, however, it is still registered inside the disabled distribution and thus is not vulnerable to subdomain takeover. The correct way to determine whether an alternate domain

is registered inside some distribution is to create a new distribution and set the alternate domain name. If the registration process does not throw an error, the custom domain is vulnerable to subdomain takeover. Figure 5.5 shows the error that is presented after the user tries to register the alternate domain name which is already present in some other CloudFront distribution.
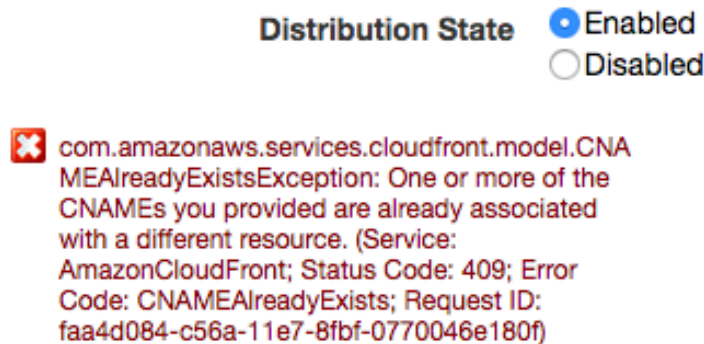

Figure 5.5: An error message shown in AWS portal during distribution creation. If an alternate domain name already exists in some other distribution, a user is presented with this message.

In section 5.3, a way of automating the detection of possible subdomain takeovers in CloudFront is presented.

5.2.2 Other

As presented in case of CloudFront, subdomain takeover is possible even on cloud services which do not have its base domain available for registration. However, since cloud services provide a way of specifying alternate domain names (CNAME records), the possibility of subdomain takeover is still present. This section provides a quick overview of other cloud services which work very similarly to CloudFront (virtual hosting architecture). Appendix C presents error messages for the non-existing alternate domain name on the following cloud services.

- **Amazon S3**
  Amazon S3 was briefly mentioned in subsection 5.2.1. The default base domain used to access the bucket is not always the same and depends on the AWS region that is used. The full list of Amazon S3 base domains is available in AWS documentation [4]. Similarly to CloudFront, Amazon S3 allows specifying the alternate (custom) domain name to access the bucket's content [Ama17a].

- **Heroku**
  Heroku is a Platform-as-a-Service provider which enables deployment of an application using simple workflow. Since access to the application is needed, Heroku

---

4. https://docs.aws.amazon.com/general/latest/gr/rande.html#s3_region

exposes the application using subdomain formed on *herokuapp.com*. However, it is also possible to specify the custom domain name to access the deployed application [Her17].

- **Shopify**
  Shopify provides a way of creating and customizing e-commerce stores in the cloud. The default subdomain to access the store is created on *myshopify.com*. As services described before, Shopify allows specifying alternate domain names [Sho17]. Noteworthy is that Shopify verifies correct CNAME record configuration. However, this verification is not domain ownership verification. Shopify only checks for correct CNAME record that is present in alternate domain's DNS zone. This verification, therefore, does not prevent subdomain takeovers.

- **GitHub**
  GitHub is a version control repository for Git. GitHub also allows free web hosting using their *GitHub Pages* project. This web hosting is usually used for project's documentation, technical blogs, or supporting web pages to open-source projects. GitHub Pages supports custom domain name in addition to default domain name under *github.io* [Git17].

- **WP Engine**
  WP Engine is hosted provider for Wordpress. Similarly to other cloud services, WP Engine provides default domain name under *wpengine.com* but custom domain name can also be specified [WP 17].

- **Microsoft Azure**
  Microsoft Azure is a bigger cloud provider, similar to AWS. It is different compared to the cloud services mentioned above in that it does not provide a virtual hosting architecture. Simply put, for each cloud service, Azure creates own virtual machine with own IP address. Therefore the mapping between a domain name and IP address is unambiguous (one-to-one mapping). Noteworthy is that since this is not a regular virtual hosting setup, configuring CNAME record does not necessary have to be explicitly defined in the resource settings. Azure provides multiple cloud services but the ones discussed in this thesis have default domains of *cloudapp.net* and *azurewebsites.net*. Its documentation describes setting the link between the domain name and Azure resource using A or CNAME records (pointing to one of the two domains mentioned previously). An interesting observation is that for A records, Azure does a domain ownership verification using TXT records [LS17]. However, it is not the case for CNAME record and subdomain takeover is therefore possible even in the case of Microsoft Azure.

5.3 FDNS for subdomain takeover detection

FDNS can be used to show the prevalence of subdomain takeover across the Internet. Because FDNS already contains resolved CNAME records, it is pretty straightforward to automate scanning for subdomain takeover across the Internet. For this purpose, a custom scanning tool was developed. This section explains its design and results.

The tool is fully written in Python and works only for CNAME records. It takes CNAME records extracted from FDNS as an input and gives the ones vulnerable to subdomain takeover output. Note however that the input does not necessarily need come from FDNS. Any input provided that uses FDNS syntax is supported.

The input is expected to be one JSON per line in the following (FDNS) format:

```
{"timestamp":"1509821526","name":"sub.example.com",
"type":"cname","value":"sub.example1.com"}
```

Where *timestamp* and *type* fields are optional. The tool, in turn, produces output in the following format (again, one JSON per line):

```
{"domain":"sub.example.com","cname":"sub.example1.com"}
```

The tool is designed to provide a plugin system. Plugins represent verification modules for an individual cloud service. Based on a method that a particular cloud service implements error handling for non-existing resources, there are two types of verification plugins:

- **HTTP verificator**
  These are the plugins which require an HTTP request to the cloud service to determine whether the domain name is vulnerable to subdomain takeover or not. As Appendix C shows, every cloud service has its unique pattern for presenting non-existing alternate domain name. HTTP verificator is checking the presence of strings in HTTP response like *this app does not exist* in order to verify subdomain takeover. Noteworthy is that the destination of HTTP request is IP address behind the canonical domain name. However, HTTP *Host* header needs to be set to the source domain name. This subtle difference is significant since without it almost all HTTP verificators provide false results.

- **DNS verificator**
  These are the plugins which require only DNS request (check for NXDOMAIN) to verify the subdomain takeover. Microsoft Azure is an example of service where DNS verificators are usable.

The correct plugin is chosen based on regular expression used for the canonical domain name (e.g., domain ending with *cloudfront.net* is verified using CloudFront verificator). All cloud services mentioned in the previous sections are supported. Complete list of available plugins (verificators) is provided in Appendix A.

The tool also contains an ability to verify CNAME records in which canonical domain name is regular Internet domain. The domain availability checking is done using *CheckDomainAvailability* API provided by AWS [5]. However, it needs to be taken into account that AWS is likely to have rate limiting for this API enabled.
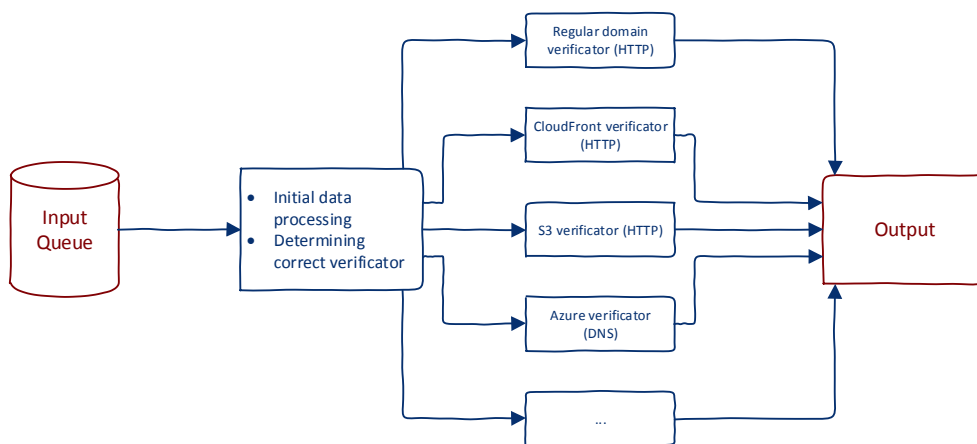

Figure 5.6: Simplified illustration of the tool's design.

Chain of CNAME records. In some instances, CNAME records might form CNAME record chains. Let's have the domain *sub.example.com* which has a CNAME record to *sub.example1.com*. If in turn, *sub.example1.com* has a CNAME record to *sub.example2.com* a three-way chain is formed:

```
sub.example.com -> sub.example1.com -> sub.example2.com
```

In such cases, when the base domain of last domain in the chain (*example2.com*) is available for registration both *sub.example1.com* and *sub.example.com* are affected. Fortunately, FDNS implicitly contains all CNAME references in the chain. For a chain given above, even though there is no direct CNAME record from *sub.example.com* to *sub.example2.com*, FDNS contains this record. Therefore, no direct changes need to be made to the automation tool to support CNAME record chains in FDNS.

Concurrency. Since FDNS currently (November 2017) contains around 18 million CNAME records, the processing of such data has to be efficient. The main bottleneck of the automation tool is network traffic (while performing domain verifications), i.e., the process is *I/O bound*. Dividing the process into multiple cores using *parallelism* would not

---

5. https://docs.aws.amazon.com/Route53/latest/APIReference/API_domains_CheckDomainAvailability.html

necessarily speed up the execution since CPU power is not the main bottleneck in this case. That is why concurrency using *Gevent* was integrated into the tool.

Gevent is working with a notion of *greenlets* [Bil15]. Greenlets are lightweight execution units (*coroutines*) similar to operating system processes. The main difference is that greenlets are scheduled by gevent, not the operating system itself. The idea of gevent and greenlets is to provide asynchronous execution. In other words, multiple tasks (greenlets) are started at once, and their result is processed in the style of *first-come-first-served*. This saves time for I/O bound processes since multiple network connections can be opened at the beginning and the results are processed in order as responses came. A normal synchronous processing, on the other hand, opens one connection, processes its response and only after that opens another one. Figure 5.7 illustrates the difference between synchronous and asynchronous execution of network related tasks.
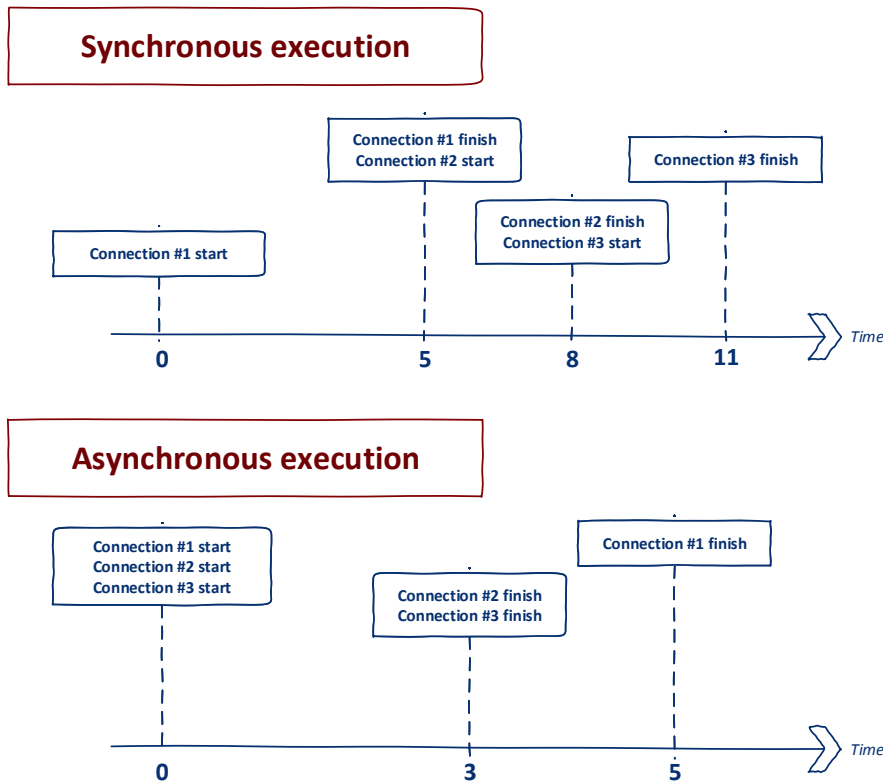

Figure 5.7: Difference between synchronous and asynchronous execution. Connection #1 takes five execution units, connections #2 and #3 take three execution units. Note that total time required for these connections in synchronous execution is 11 as opposed to 5 in asynchronous execution.

The tool presented in this chapter uses *gevent.Pool* [6]. This construct allows handling multiple greenlets as a group. The idea is to have multiple connections open at any given

---

6. http://www.gevent.org/gevent.pool.html

time. After response for one of the connections is returned, its data is processed, and it is automatically replaced by another connection to keep the fixed size of a group.

There are at least three open-source tools available for subdomain takeover verification: *subjack* [7], *HostileSubBruteforcer* [8] and *xcname* [9]. Even though the above projects contain an ability to verify a similar set of cloud services, they are not optimized for a large number of records such as those in FDNS. Therefore, custom automation tool was needed. The tool presented in this chapter provides rather simple architecture. When a new cloud service without proper domain verification is identified, only a new verification plugin is required to support this cloud service.

5.3.1 Results

Scanning all CNAME records in FDNS (around 30 million records) put notable stress on domain availability checker in AWS. The decision was made to scan only CNAME records in which canonical name is owned by one of the supported cloud providers. This might seem like an overly limited scan, however, cloud providers make the majority of CNAME records in FDNS.

In FDNS dataset from November 3rd, 2017, **12 888** source domain names in CNAME records were vulnerable to subdomain takeover. Figure 5.8 shows the distribution of cloud providers in canonical domain names in affected records. Note that scanning all CNAME records in FDNS would certainly lead to more vulnerable source domain names.
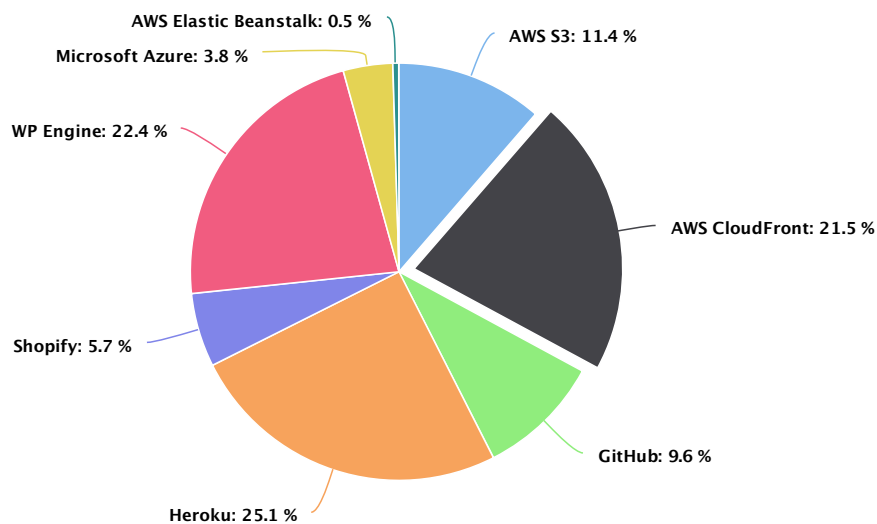

Figure 5.8: Distribution of cloud services in the result set.

---

7. https://github.com/haccer/subjack
8. https://github.com/nahamsec/HostileSubBruteforcer
9. https://github.com/mandatoryprogrammer/xcname

Because of a high number of results, a prioritization strategy for domain names was used. *Alexa 1 million* is de-facto a standard dataset which is used in academic papers to obtain a list of one million most popular websites/domains on the Internet. Since Alexa is no longer providing an updated dataset for free, an alternative dataset called *Majestic Million* [10] is used in this section.

From the all possible subdomain takeovers found in FDNS, ten samples with highest Majestic Million rank were chosen for demonstration. A notification e-mail explaining the problem and possible mitigation strategies (see section 5.5) were sent to each domain's administrator. Appendix D shows such notification e-mail.

Domain name	Cloud provider	Rank	Mitigated?
REDACTED.microsoft.com	Azure	5	YES
REDACTED.mit.edu	Github	88	NO
REDACTED.nasa.gov	CloudFront	103	YES
REDACTED.skype.com	Azure	202	YES
REDACTED.intel.com	CloudFront	212	YES
REDACTED.cmu.edu	Heroku	349	YES
REDACTED.ieee.org	CloudFront	374	YES
REDACTED.gotomeeting.com	CloudFront	1428	YES
REDACTED.avira.com	CloudFront	1536	YES
REDACTED.ryanair.com	AWS S3	1548	NO

Table 5.1: Domain names vulnerable to subdomain takeover. Column *Rank* represents position of base domain in Majestic Million list from November 2017.

Because of legal considerations, this thesis is not listing the specific domains that were affected. Some of these domains have not yet mitigated the subdomain takeover, and others have not permitted disclosing the problem.

5.4 Implications

After an explanation of subdomain takeover and its demonstration using FDNS, this section discusses real-world implications that exist after the attacker takes over some legitimate domain. Liu et al. provide a comprehensive explanation of consequences in their paper [LHW16]. This section tries to extend their explanation using different scenarios and examples.

Phishing

Attackers are often using *typosquatting* [Szu+14] or so-called *Doppelganger domains* [GK11] to mimic the legitimate domain/website for phishing purposes. Typosquatting is a technique of registering domain names which look similar to some legitimate domain name.

---

10. https://majestic.com/reports/majestic-million

For instance given *google.com*, one example of typosquatting domain might be *g00gle.com* (notice the "zero" instead of "o"). Such domain name appears identical to the original one. A doppelganger domain is similar to typosquatting domain. It is a domain which is missing "." (dot) in a domain name. For example, an instance of Doppelganger domain for *mail.google.com* is *mailgoogle.com* (notice the missing dot). When the content on these domain matches branding and content of the original website, users are not able to tell the difference and are more likely to be tricked by an attacker (e.g., for credential harvesting or financial fraud).

After an attacker takes over some legitimate domain name, it is almost impossible for a regular user to tell whether the content on the domain is provided by a legitimate party or an attacker. Let's take for instance a random bank. If one of the bank's subdomains is vulnerable to subdomain takeover, an attacker can create an HTML form which mimics the login form to the bank's internet banking system. Then, an attacker can run spear phishing or mass phishing campaign asking users to log in to and change their passwords. At this stage, the passwords are captured by an attacker who is in control of domain in question. The URL provided in the phishing e-mail is a legitimate subdomain of a bank. Therefore users are not aware of something malicious going on. Spam filters and other security measurements are also less likely to trigger the e-mail as spam or malicious because it contains domain names with higher trust.

This attack can be enhanced by generating a valid SSL certificate. Certificate authorities such as *Let's Encrypt* allow automatic verification of domain ownership by content verification (see Figure 5.9). That is, if there is a specific content placed on specific URL path, Let's Encrypt will approve the issuance of a certificate for a given domain [Let]. Since an attacker has full control over the content of the domain which is vulnerable to subdomain takeover, this verification can be done in a matter of minutes. Therefore attackers are also able to generate SSL certificate for such domain which only lowers the suspicion of a phishing attack.
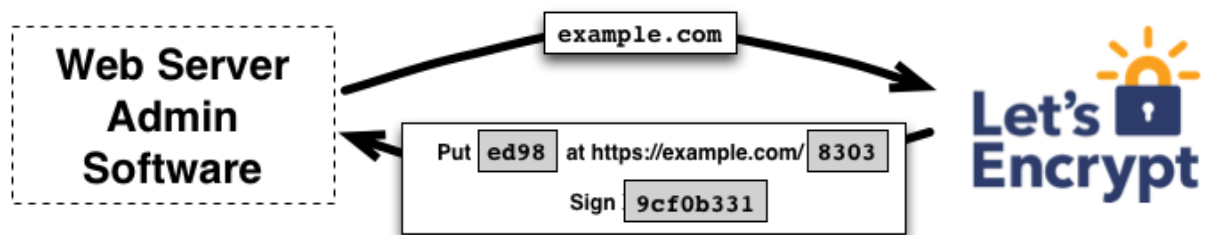

Figure 5.9: Simplified process of Let's Encrypt verification [Let].

A different problem with (CNAME) subdomain takeover is that it also allows receiving and sending e-mails from the affected domain. CNAME records delegates every DNS resource record, MX not being an exception. This means that the mail server can be set up on a taken domain and the attacker receives all e-mails targeted to this domain. E-mail sending is no difference. *SPF*, *DKIM*, and *DMARC* records can be configured in TXT records (TXT is delegated too). Having capability of sending and receiving e-mails is more critical for spear phishing attacks where a reply to the original e-mail is usually required

from the target. In 2015, Ubiquiti Network lost \$46 million because of phishing e-mail spoofing CEO [Kre15]. The attacker vector included spear phishing e-mail with a changed *reply-to* address to receive further responses from the victim. Once attacker takes over the legitimate domain, the *reply-to* address does not need to be changed, and the attacker has higher chances of success.

One of the real-world examples of subdomain takeover was an incident which occurred to one of Donald Trump's domains. A hacker was able to take *secure2.donaldjtrump.com* at the beginning of 2017 [Bis17]. Although the hacker has not used this website for phishing purposes, it might have been used to cause harm to Donald Trump's presidential campaign.

Another example is bug bounty report created in 2014 by Frans Rosén [Ros14]. In this report, the domain name *media.vine.com* (popular video website) was vulnerable to subdomain takeover. Possible scenarios might have been similar to the one explained with a bank above: an attacker might setup HTML website similar to *vine.com* and harvest user credentials.

Note however that some of the scenarios explained above will work only on the canonical domain names which are regular Internet domains. If the domain points to AWS Cloud-Front, there is, for instance, no chance of receiving e-mails to this domain.

Cross-site scripting

Cookies are a way of storing temporary information in web browsers [Bar11]. Since HTTP is a stateless protocol, cookies are used to track session information across different requests to the server. For instance, when a user logs in to a website, a unique cookie (called *session cookie*) is stored in user's web browser. This session cookie is sent with every upcoming HTTP request. The server can send HTTP requests to the particular user using a session cookie. Having access to session cookie often means having the authorization of the user in the context of web application which assigned this session cookie. Browsers, therefore, restrict cookies in a several ways:

- **Same domain policy**
  The cookie issued by one particular domain can be accessed only by web server residing on that same domain. One exception to this policy is explained further below.

- **HttpOnly cookie**
  Cookies can by default be accessed by Javascript code running in the context of the website which created the cookies. Javascript can read, update, and delete the cookies. *HttpOnly* cookie flag (set by the web server) indicates that the particular cookie cannot be accessed by Javascript code. The only way to get it is through HTTP request and response headers.

- **Secure cookie**
  When the cookie has the *Secure* flag set by the web server, it can be communicated back to the web server only if HTTPS is used.

If the domain is vulnerable to subdomain takeover, an attacker can gather cookies issued by that domain in the past just by tricking users into visiting that website. HttpOnly and Secure flags will not help since the cookie is not being accessed using Javascript and SSL certificate can be easily generated for the taken domain.

Cookies can also be shared across subdomains. This usually happens when the website uses cookie-based *Single sign-on (SSO)* system. Using SSO, a user can log in using one subdomain and share the same session token across a wide range of subdomains. The syntax for setting a regular cookie is the following:

```
HTTP/1.1 200 OK
Set-Cookie: name=value
```

If this cookie is issued by web server residing on *example.com*, only this server can access this cookie later on. However, the cookie can be issued for wildcard domain (for the reasons explained above) in the following manner:

```
HTTP/1.1 200 OK
Set-Cookie: name=value; domain=example.com
```

The cookie will be included in HTTP requests to *example.com* but also to any other subdomain such as *subdomain.example.com*. This behavior creates a possibility of high severity attacks using subdomain takeover. Suppose that some particular domain is using session cookies for wildcard domain. If there is one subdomain vulnerable to subdomain takeover, the only thing for gathering user's session token is to trick him or her into visiting the vulnerable subdomain. The session cookie is automatically sent with the first HTTP request.

This technique was explained in bug bounty report by Arne Swinnen [Swi16]. The report explains the problem with one of the *Ubiquiti Networks* subdomains (*ping.ubnt.com*). This subdomain was vulnerable to subdomain takeover, pointing to unclaimed AWS CloudFront distribution. Since Ubiquiti Networks is using SSO with wildcard session cookies, all users visiting *ping.ubnt.com* could have their session cookies stolen. Even though this domain is pointing to AWS CloudFront, CloudFront distribution settings allow logging cookies with each request. Therefore the scenario with extracting session cookies is entirely possible even with subdomains pointing to AWS CloudFront. In 2017, Swinnen also demonstrated similar attack vector against Uber's SSO system [Swi17].

The behavior explained above is not limited to cookies. Since Javascript scripts have full control over the websites, they are run on, having ability to replace such scripts on the legitimate website might lead to catastrophic consequences. Suppose that website is using Javascript code from the external provider using *script* tag and *src* attribute. When the domain of external provider expires, the browser fails silently, i.e., it will not trigger

any alerts visible to regular users. If the external code is not doing any crucial stuff (e.g., it is used only for tracking) such external provider might stay on the website for a long period. An attacker can take over this expired domain, match the URL path of provided Javascript code and thus gain control over every visitor that visits the original website.

In 2017, an attacker gained control over DNS record of cryptocurrency mining company *CoinHive* [Osb17]. CoinHive provides Javascript files that web developers can include on their websites. These files are used for cryptocurrency mining purposes – for the time that user stays on a website, her browser is used for mining in the meantime. This technique is being utilized as a replacement for a regular advertisement for generating revenue. Attackers were able to change DNS record for at least 6 hours because of leaked passwords of CoinHive admins. Changed DNS records were used to provide updated versions of CoinHive's Javascript files which provided mining revenue directly to attackers. Although attackers did not use subdomain takeover as an attack vector, this incident provides a red alert that such attacks are indeed happening.

There is, however, one way of protecting the integrity of Javascript files in a browser. *Subresource Integrity* was proposed as a mechanism to include cryptographic hash as an attribute *integrity* to *script* tag in HTML5 [Akh+16]. When the provided cryptographic hash does not match the download file, the browser refuses to execute it.

5.5   Mitigation

The mitigation strategies for domain names already vulnerable to subdomain takeover are rather straightforward:

- **Remove the affected DNS record**
  The simplest solution is to remove the affected record from the DNS zone. This step is usually used if the organization determines that the affected source domain name is no longer needed.

- **Claim the canonical domain name**
  This means registering the resource in particular cloud provider or in case of a regular Internet domain, buying the expired domain back.

To prevent subdomain takeover in the future, organizations should change the process of creating and destructing resources in their infrastructure. In case of resource creation, the DNS record creation has to be the *last step* of this process. This condition prevents DNS record to be pointing to a non-existing domain at any point in time. For resource destruction, the opposite holds: DNS record needs to be removed as the *first step* in this process.

Mitigation strategy for cloud providers should be considered as well. As was seen throughout this chapter, some cloud services are not verifying the domain ownership. The reason behind this is primarily convenience. Cloud provider is not introducing any vulnerability by not verifying ownership of a source domain name. It is therefore up to the user to monitor its own DNS records.

These are two examples of cloud providers which include domain ownership verification as part of CNAME delegation:

- **Google Cloud Platform (GCP)**
  GCP requires domain ownership verification using TXT records [Goo17a]. Google generates a unique string which an administrator needs to put into TXT record after the alternate domain name is configured. Google then queries the domain name in question to verify whether this string is present in the DNS response.

- **Squarespace**
  Similarly to GCP, Squarespace requires the domain ownership verification using additional CNAME record [Squ17].

Both of the examples above prevent subdomain takeover. Because the attacker does not have an access to full DNS zone of the source domain name, she would not be able to put specific DNS record there. These cloud providers refuse to handle CNAME delegation without a proper domain ownership verification. Noteworthy is that in spite of domain verification present, Evgeny Morozov in 2017 demonstrated that domain verification can be bypassed under some circumstances [Mor17]. He used a rather simple technique of DNS spoofing which consists of proactively sending DNS responses to the verification server.

6 Conclusions

This thesis provided an analysis of domain names in cybersecurity context. Two primary topics were covered: *domain correlation* and *subdomain takeover.*

Firstly, the comparison of public DNS dataset was performed in order to select the appropriate dataset. Even though there are multiple public DNS datasets, the Rapid7 Forward DNS dataset (FDNS) was chosen, described, and widely analyzed. The FDNS served as a foundation for topics explained in *chapter 4* and *chapter 5.*

Vertical domain correlation and *horizontal domain correlation* were defined in *chapter 4.* Both of these methods were extended with novel techniques which use FDNS as the main data source. Comparison between known techniques and techniques using FDNS are available in their particular sections. The comparison results show that combining already known techniques with FDNS can lead to better domain correlation results. This thesis also provides easy-to-use scripts for every demonstrated technique. These techniques should help organizations to see their public exposure from the eyes of cyber adversaries.

FDNS also served as a foundation for *subdomain takeover* analysis described in *chapter 5.* Subdomain takeover is still an evolving topic in cybersecurity. This thesis extended already published research with new real-world examples and its implications. The section 5.4 showed that attacks involving subdomain takeover are not purely theoretical. As part of the analysis, several vulnerable domains owned by high-profile organizations were identified. These organizations were proactively informed about this issue with possible mitigation strategies.

Although mitigation of subdomain takeover is rather a simple process, it often involves human-factor. With the further utilization of cloud services, subdomain takeover vulnerabilities will continue to grow. There is still an open question about how the cloud providers should approach domain ownership verification. Indeed, domain ownership verification process complicates a cloud service registration process. However, correct domain ownership verification processes also mitigate subdomain takeover. It will be interesting to see whether cloud providers will start to adopt it or not.

As both *domain correlation* and *subdomain takeover* results demonstrated, FDNS is a powerful dataset which provides a large amount of information. Up to this date (November 2017), there are not many publicly available projects utilizing FDNS. Hopefully, this thesis will serve as a pioneering example that provides new perspectives into doing DNS related researches.

Bibliography

[Akh+16]   D. Akhawe, F. Braun, F. Marier, and J. Weinberger. (2016). Subresource Integrity, [Online]. Available: https://www.w3.org/TR/2016/REC-SRI-20160623/ (visited on 11/01/2017).

[Ama17a]   Amazon Web Services. (2017). Virtual Hosting of Buckets, [Online]. Available: https://docs.aws.amazon.com/AmazonS3/latest/dev/VirtualHosting.html (visited on 11/07/2017).

[Ama17b]   Amazon Web Services. (2017). What is Amazon Cloudfront?, [Online]. Available: https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/Introduction.html (visited on 11/07/2017).

[Ama17c]   Amazon Web Services. (2017). Using Alternate Domain Names (CNAMEs), [Online]. Available: https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/CNAMEs.html (visited on 11/07/2017).

[Ant+17]   M. Antonakakis, T. April, M. Bailey, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, D. Menscher, C. Seaman, N. Sullivan, *et al.*, "Understanding the Mirai Botnet", 2017.

[Bar11]   A. Barth, "HTTP state management mechanism", 2011.

[Bil15]   D. Bilenko. (2015). Introduction - gevent 1.3.0.dev0 documentation, [Online]. Available: http://www.gevent.org/intro.html (visited on 11/06/2017).

[Bis17]   D. Bisson. (2017). Hacker defaces Donald Trump fundraising site via subdomain takeover attack, [Online]. Available: https://www.grahamcluley.com/hacker-defaces-donald-trump-fundraising-site-via-subdomain-takeover-attack/ (visited on 10/31/2017).

[Bla+06]   S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, and T. Wright, "Transport layer security (TLS) extensions", Tech. Rep., 2006.

[Blo08]   I. J. Block, "Hidden Whois and Infringing Domain Names: Making the Case for Registrar Liability", *U. Chi. Legal F.*, 2008.

[Bre17]   B. Brenner. (2017). Thousands of MongoDB databases compromised and held to ransom, [Online]. Available: https://nakedsecurity.sophos.com/2017/01/11/thousands-of-mongodb-databases-compromised-and-held-to-ransom/ (visited on 10/31/2017).

[Bry16]   M. Bryant. (2016). The .io error – Taking Control of All .io Domains With a Targeted Registration, [Online]. Available: https://thehackerblog.com/the-io-error-taking-control-of-all-io-domains-with-a-targeted-registration/index.html (visited on 11/02/2017).

[Bry17]   M. Bryant. (2017). The International Incident – Gaining Control of a .int Domain Name With DNS Trickery, [Online]. Available: https://thehackerblog.com/the-international-incident-gaining-control-of-a-int-domain-name-with-dns-trickery/index.html (visited on 11/02/2017).

[Cen]   Censys Team, *Internet-Wide Scan Data Repository*. [Online]. Available: `https://scans.io/` (visited on 11/11/2017).

[CR13]   R. Chandramouli and S. Rose, "Secure domain name system (DNS) deployment guide", *NIST Special Publication*, 2013.

[DBH14]   Z. Durumeric, M. Bailey, and J. A. Halderman, "An Internet-Wide View of Internet-Wide Scanning", in *USENIX Security Symposium*, 2014.

[DR17]   M. Dooley and T. Rooney, *DNS Security Management*. Wiley-IEEE Press, 2017, ISBN: 9781119328278.

[Dur+15]   Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman, "A Search Engine Backed by Internet-Wide Scanning", in *Proceedings of the 22nd ACM Conference on Computer and Communications Security*, 2015.

[Fan+15]   L. Fang, W. Bailing, H. Junheng, S. Yushan, and W. Yuliang, "A proactive discovery and filtering solution on phishing websites", in *Big Data (Big Data), 2015 IEEE International Conference on*, IEEE, 2015, pp. 2348–2355.

[Git17]   GitHub. (2017). Using a custom domain with Github Pages, [Online]. Available: `https://help.github.com/articles/using-a-custom-domain-with-github-pages/` (visited on 11/11/2017).

[GK11]   G. Gee and P. Kim. (2011). Doppelganger Domains, [Online]. Available: `http://godaigroup.net/wp-content/uploads/doppelganger/Doppelganger.Domains.pdf` (visited on 10/31/2017).

[GM15]   O. Gudmundsson and M. Majkowski. (2015). Deprecating the DNS ANY meta-query type, [Online]. Available: `https://blog.cloudflare.com/deprecating-dns-any-meta-query-type/` (visited on 10/09/2017).

[Goo12]   D. Goodin. (2012). Hackers expose 453,000 credentials allegedly taken from Yahoo service (Updated), [Online]. Available: `https://arstechnica.com/information-technology/2012/07/yahoo-service-hacked/` (visited on 11/11/2017).

[Goo17a]   Google. (2017). Mapping Custom Domains, [Online]. Available: `https://cloud.google.com/appengine/docs/standard/python/mapping-custom-domains` (visited on 11/07/2017).

[Goo17b]   Google. (2017). Refine web searches, [Online]. Available: `https://support.google.com/websearch/answer/2466433?hl=en` (visited on 10/08/2017).

[GW95]   J. Gargano and K. Weiss, "Whois and Network Information Lookup Service Whois++", Tech. Rep., 1995. [Online]. Available: `https://tools.ietf.org/html/rfc1834`.

[Har17]   J. Hart. (2017). Forward DNS, [Online]. Available: `https://github.com/rapid7/sonar/wiki/Forward-DNS` (visited on 10/07/2017).

[Her17]   Heroku. (2017). Custom Domain Names for Apps, [Online]. Available: `https://devcenter.heroku.com/articles/custom-domains` (visited on 11/07/2017).

[HHH17]   N. Huq, S. Hilt, and N. Hellberg, "US Cities Exposed: Industries and ICS", TrendMicro, Tech. Rep., 2017.

[Imp17]   Imperva. (2017). What is a CDN, [Online]. Available: https://www.inca psula.com/cdn-guide/what-is-cdn-how-it-works.html (visited on 11/07/2017).

[Int15]   Internetwache.org. (2015). Scanning Alexa's Top 1M for AXFR, [Online]. Available: https://en.internetwache.org/scanning-alexas-top-1m-for-axfr-29-03-2015/ (visited on 10/14/2017).

[Ker16]   C. Kerschbaumer, "Enforcing content security by default within Web browsers", in *Cybersecurity Development (SecDev), IEEE*, 2016, pp. 101–106.

[Kha+15]   M. T. Khan, X. Huo, Z. Li, and C. Kanich, "Every second counts: Quantifying the negative externalities of cybercrime via typosquatting", in *Security and Privacy (SP), 2015 IEEE Symposium on*, IEEE, 2015, pp. 135–150.

[KKvE16]   M. Korczyński, M. Król, and M. van Eeten, "Zone Poisoning: The How and Where of Non-Secure DNS Dynamic Updates", in *Proceedings of the 2016 ACM on Internet Measurement Conference*, ACM, 2016, pp. 271–278.

[Kre15]   B. Krebs. (2015). Tech Firm Ubiquiti Suffers $46M Cyberheist, [Online]. Available: https://krebsonsecurity.com/2015/08/tech-firm-ubiquiti-suffers-46m-cyberheist/ (visited on 11/01/2017).

[Kre17]   I. Kreymer. (2017). CDX Server API, [Online]. Available: https://github.com/ikreymer/pywb/wiki/CDX-Server-API (visited on 10/14/2017).

[Küh15]   M. Kühne, "Introduction to IP Addressing and Regional Internet Registries", Tech. Rep., 2015.

[Let]   Let's Encrypt, *How it works - Let's encrypt*. [Online]. Available: https://letsencrypt.org/how-it-works/ (visited on 11/01/2017).

[Lev12]   P. Levis, "The Collateral Damage of Internet Censorship by DNS Injection", *ACM SIGCOMM CCR*, vol. 42, no. 3, 2012.

[LHW16]   D. Liu, S. Hao, and H. Wang, "All Your DNS Records Point to Us: Understanding the Security Threats of Dangling DNS Records", in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2016, pp. 1414–1425.

[LLK13]   B. Laurie, A. Langley, and E. Kasper, "Certificate transparency", Tech. Rep., 2013.

[LS16]   A. Liska and G. Stowe, *DNS Security: Defending the Domain Name System*. Syngress, 2016, ISBN: 9780128033067.

[LS17]   C. Lin and B. Surowiec. (2017). Map an existing custom DNS name to Azure Web Apps, [Online]. Available: https://docs.microsoft.com/en-us/azure/app-service/app-service-web-tutorial-custom-domain (visited on 11/11/2017).

[LW07]   L. Lancor and R. Workman, "Using Google hacking to enhance defense strategies", in *ACM SIGCSE Bulletin*, ACM, vol. 39, 2007, pp. 491–495.

[Mie17]   D. Miessler, *SecLists*, 2017. [Online]. Available: `https://github.com/daniel miessler/SecLists`.

[Mo015]   H. D. Moore. (2015). Reverse DNS, [Online]. Available: `https://github.com/rapid7/sonar/wiki/Reverse-DNS` (visited on 10/31/2017).

[Moo15]   H. D. Moore. (2015). More SSL Certificates, [Online]. Available: `https://github.com/rapid7/sonar/wiki/More-SSL-Certificates` (visited on 10/31/2017).

[Mor17]   E. Morozov. (2017). Bypassing domain control verification with DNS response spoofing, [Online]. Available: `https://labs.detectify.com/2017/09/11/guest-blog-bypassing-domain-control-verification-with-dns-response-spoofing/` (visited on 11/07/2017).

[Moz]   Mozilla Foundation, *Public Suffix List*. [Online]. Available: `https://publicsuffix.org/` (visited on 11/16/2017).

[MSK12]   S. McClure, J. Scambray, and G. Kurtz, *Hacking Exposed 7: Network Security Secrets and Solutions*. McGraw-Hill Education, 2012, ISBN: 9780071613743.

[Nag17]   S. Nagel. (2017). September 2017 Crawl Archive Now Available, [Online]. Available: `https://commoncrawl.org/2017/09/september-2017-crawl-archive-now-available/` (visited on 10/11/2017).

[Osb17]   C. Osborne. (2017). Hackers hijack Coinhive cryptocurrency miner through an old password, [Online]. Available: `http://www.zdnet.com/article/hackers-hijack-coinhive-dns-server-through-an-old-password/` (visited on 10/31/2017).

[Ros14]   F. Rosén. (2014). URGENT - Subdomain Takeover on media.vine.co due to unclaimed domain pointing to AWS, [Online]. Available: `https://hackerone.com/reports/32825` (visited on 10/31/2017).

[Sag14]   T. Sager, "Killing Advanced Threats in Their Tracks: An Intelligent Approach to Attack Prevention", SANS Institute, Tech. Rep., 2014.

[Sho]   Shodan, *Shodan*. [Online]. Available: `https://www.shodan.io/` (visited on 11/11/2017).

[Sho17]   Shopify. (2017). Setting up your existing domain, [Online]. Available: `https://help.shopify.com/manual/domains/connecting-existing-domains/setting-up-your-domain` (visited on 11/07/2017).

[Squ17]   Squarespace. (2017). Connecting a domain to your Squarespace site, [Online]. Available: `https://support.squarespace.com/hc/en-us/articles/205812378-Connecting-a-domain-to-your-Squarespace-site` (visited on 11/07/2017).

[Swi16]   A. Swinnen. (2016). Authentication bypass on sso.ubnt.com via subdomain takeover of ping.ubnt.com, [Online]. Available: `https://hackerone.com/reports/172137` (visited on 10/31/2017).

[Swi17]   A. Swinnen. (2017). Authentication bypass on Uber's Single Sign-On via sub-domain takeover, [Online]. Available: `https://www.arneswinnen.net/2017/06/authentication-bypass-on-ubers-sso-via-subdomain-takeover/` (visited on 10/31/2017).

[Szu+14]   J. Szurdi, B. Kocso, G. Cseh, J. Spring, M. Felegyhazi, and C. Kanich, "The Long" Taile" of Typosquatting Domain Names", in *USENIX Security Symposium*, 2014, pp. 191–206.

[The17]   The Apache Software Foundation. (2017). Apache Virtual Host documentation, [Online]. Available: `https://httpd.apache.org/docs/current/vhosts/` (visited on 10/14/2017).

[Wag+12]   C. Wagner, J. François, R. State, T. Engel, G. Wagener, and A. Dulaunoy, "SDBF: Smart DNS brute-forcer", in *Network Operations and Management Symposium (NOMS)*, IEEE, 2012, pp. 1001–1007.

[Wei17]   K. Weins. (2017). Cloud Computing Trends: 2017 State of the Cloud Survey, [Online]. Available: `https://www.rightscale.com/blog/cloud-industry-insights/cloud-computing-trends-2017-state-cloud-survey` (visited on 11/03/2017).

[WP 17]   WP Engine. (2017). How to configure your DNS for WP Engine, [Online]. Available: `https://wpengine.com/support/wordpress-best-practice-configuring-dns-for-wp-engine/` (visited on 11/11/2017).

[Wri15]   J. Wright. (2015). How to Download a List of All Registered Domain Names, [Online]. Available: `https://jordan-wright.com/blog/2015/09/30/how-to-download-a-list-of-all-registered-domain-names/` (visited on 10/31/2017).

[Yia13]   C. Yiannis, "Modern Password Cracking: A hands-on approach to creating an optimised and versatile attack", 2013.

A Archive structure

Content of the attached archive:

```
Archive
├── active_domains.py (Filters active domains)
├── base_domain.py (Extracts base domain from the given domain name)
├── common_crawl.sh (Vertical domain correlation using Common Crawl)
├── data_analysis.sh (FDNS data analysis)
├── fdns_horizontal.sh (Horizontal domain correlation using FDNS)
├── fdns_vertical.sh (Vertical domain correlation using FDNS)
├── ip_range.sh (Horizontal domain correlation using IP ranges)
├── ns_group.py (Helper script for horizontal domain correlation)
├── san.sh (Vertical domain correlation using SubjectAltName)
├── tld_extract.py (Extracts TLD from the given domain name)
└── takeover_scan/
    ├── fdns.py (Main script which iterates through input)
    ├── requirements.txt (Required Python packages)
    └── subdomain_takeover/
        ├── __init__.py
        ├── resolver.py (DNS resolution helper)
        └── plugins/ (Verificators)
            ├── __init__.py
            ├── aws.py (AWS verificator)
            ├── azure.py (Microsoft Azure verificator)
            ├── base.py (Contain parent classes for verificators)
            ├── github.py (GitHub verificator)
            ├── heroku.py (Heroku verificator)
            ├── shopify.py (Shopify verificator)
            └── wpengine.py (WP Engine verificator)
```

B Shodan and Censys screenshots

Figure B.1: Web interface of Shodan, showing results for 74.125.206.105

Figure B.2: Web interface of Censys, showing results for 8.8.8.8

C Error messages in cloud services

ERROR

The request could not be satisfied.

Bad request.

```
Generated by cloudfront (CloudFront)
Request ID: sDIuheTmeArp89DDJVKisBA1-Ol2ueGFW8_rSIvNb2VFojbWhOEvmA==
```

Figure C.1: Non-existing alternate domain name in Amazon CloudFront.

404 Not Found

- Code: NoSuchBucket
- Message: The specified bucket does not exist
- BucketName:
- RequestId: 5E29D2B278D5F88E
- HostId: yc93yUGwBuih7IoLihIeOK9BuSiedniDWzEu24mWzCBLeaUzPJS+5drDFsitzMj4rebGK7FXTIc=

Figure C.2: Non-existing alternate domain name in Amazon S3.

Figure C.3: Non-existing alternate domain name in Heroku.
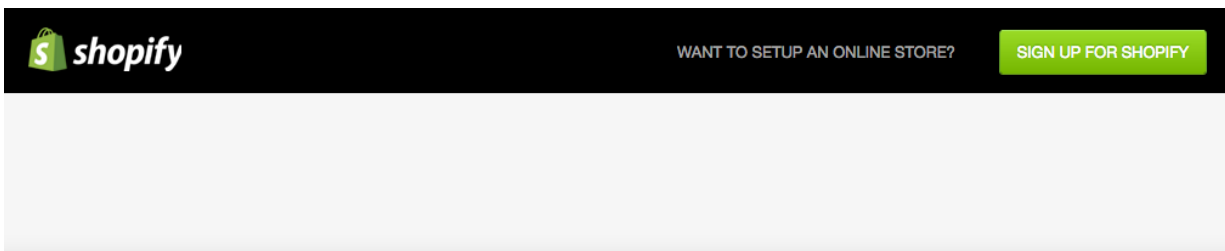
Figure C.4: Non-existing alternate domain name in Shopify.
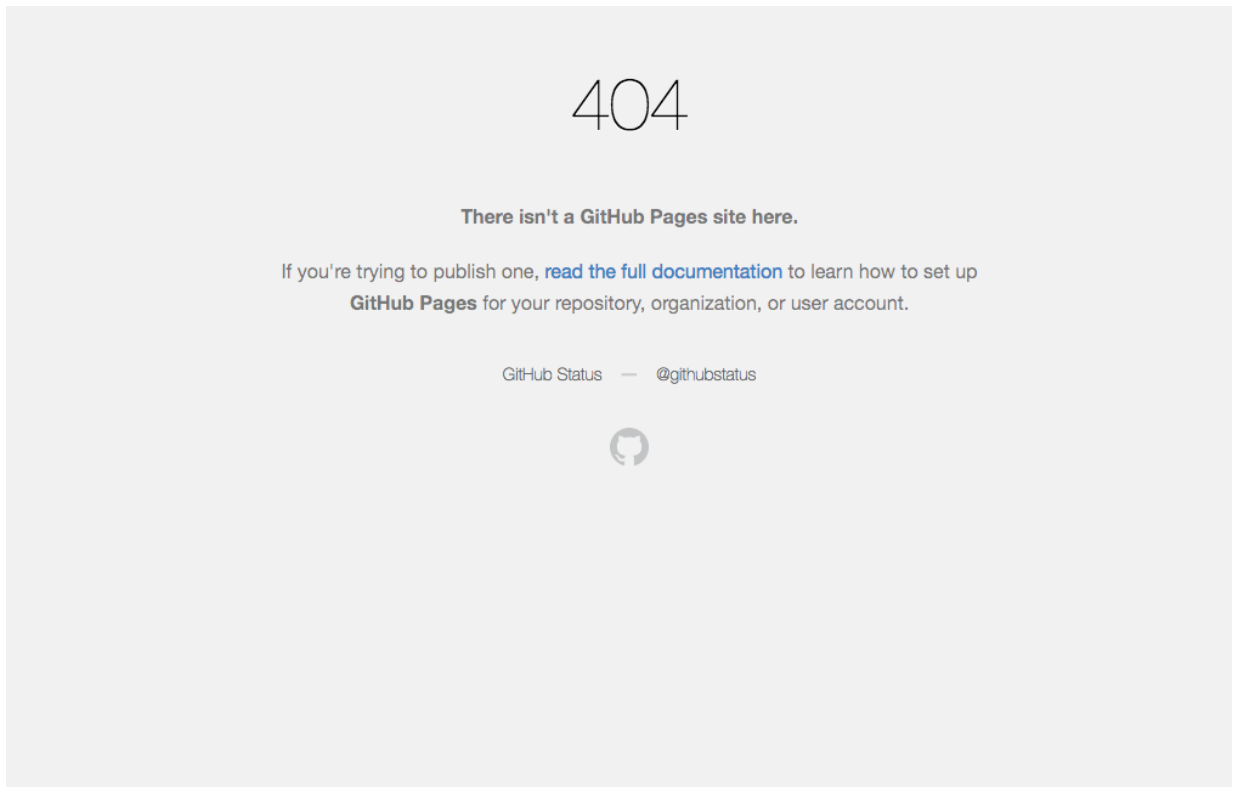
Figure C.5: Non-existing alternate domain name in GitHub.

Figure C.6: Non-existing alternate domain name in WP Engine.
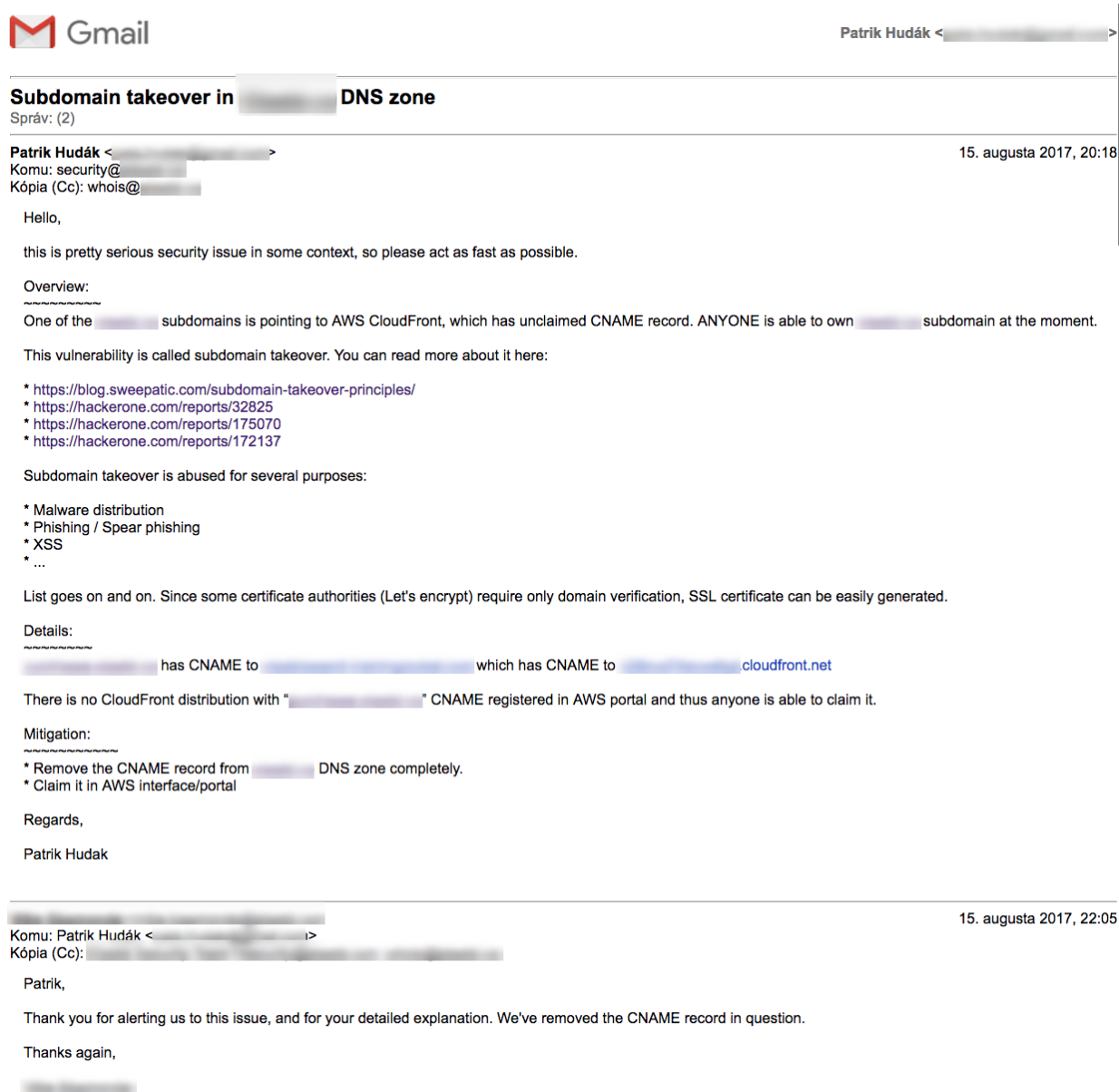
D Subdomain takeover notification e-mail

Figure D.1: Sample subdomain takeover notification e-mail. It was sent to administrators of vulnerable domain.