Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

Localised Internet Scanning Infrastructure

Jordan Myers
15323206

B.A. (Mod) Integrated Computer Science
Final Year Project April 2019
Supervisor: Dr. Stephen Farrell

Declaration

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

Signed: _____ Date: _____

Abstract

A short summary of the problem investigated, the approach taken and the key findings. This should be around 400 words, or less.

This should be on a separate page.

Acknowledgements

You should acknowledge any help that you have received (for example from technical staff), or input provided by, for example, a company.

Acronyms

CRUD	Create, Read, Update, Delete
IANA	Internet Assigned Numbers Authority
MAC	Message Authentication Code
SMTP	Simple Mail Transfer Protocol
SSH	Secure Shell
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol

Contents

1 Introduction **1**

1.1 Goal . . . 1
1.2 Motivation . . . 1
1.3 Report Outline . . . 2

2 Background & Related Work **3**

2.1 Internet Scanning . . . 3
2.1.1 ZMap & ZGrab . . . 3
2.1.2 Censys.io . . . 4
2.2 Network Scanning . . . 5
2.3 Port Scanning . . . 5
2.4 Project Background . . . 6
2.5 Data Overview . . . 7
2.6 Elastic Stack . . . 7
2.7 Transmission Control Protocol . . . 8
2.8 Application Layer Internet Protocols . . . 8
2.8.1 SSH . . . 8
2.8.2 SMTP . . . 9
2.8.3 POP3 . . . 9
2.8.4 IMAP & IMAPS . . . 9
2.8.5 HTTPS . . . 9
2.9 Transport Layer Security . . . 9
2.9.1 TLS Handshake . . . 10
2.9.2 Cipher Suite . . . 11
2.9.3 Public Key Fingerprint . . . 11
2.9.4 SHA256 . . . 11

3 Data Store **12**

3.1 Data Types and Mappings . . . 12
3.2 Index Templates . . . 13

3.3 CLI Tool . . . 14
3.3.1 Installing Elasticsearch & Kibana . . . 14
3.3.2 Extracting Subset . . . 15
3.3.3 Inserting Data . . . 16
3.3.4 Managing Indices . . . 16
3.4 Challenges . . . 17
3.4.1 Elasticsearch Crash on Bulk Insert . . . 17

4 Data Analysis **21**
4.1 Kibana . . . 21
4.1.1 Web Server and TLS Certificate . . . 21
4.1.2 Visualisations . . . 21
4.2 Dashboards . . . 22
4.2.1 Ireland Data . . . 22
4.2.2 Aviation Data Subset . . . 22
4.3 Results/Insights . . . 22

5 Evaluation **23**
5.1 Same Keys . . . 23
5.1.1 Lines of Code . . . 23
5.1.2 Run Time . . . 23
5.2 Scalability . . . 23
5.3 Usability . . . 23

6 Conclusion **24**

7 Future Work **25**

A1 Appendix **29**

List of Figures

2.1 Censys.io scanning and annotation process [1] 4
2.2 Censys.io system architecture diagram [1] 5

3.1 Command to install Elasticsearch and Kibana 14
3.2 Command to extract data subset 15
3.3 Command to insert JSON file into Elasticsearch 16
3.4 Command to view all indices 16
3.5 Command to delete index 17
3.6 Elasticsearch architecture with one cluster, two nodes, two indexes with three shards, each with one replica 19

List of Tables

2.1 Scan data used for development 7

3.1 Settings in default Elasticsearch index template used for this project. 14

1 Introduction

1.1 Goal

The goal of this project is to implement a data store and data visualisation tool for a localised internet scanning infrastructure which supports Ireland-sized internet scans. It follows on from work completed previously on surveying cryptographic public key re-use, outlined in Section 2.4.

The project aims to achieve the following goals:

- Implement a data store.
- Provide an easy way to ingest scan data.
- Allow easy data analysis and visualisation.
- Use open-source software.
- Be able to run on a machine with limited resources but also be scalable.
- Outline steps to deploy to a production environment.

1.2 Motivation

While large-scale, internet-wide scanning is a well-explored topic, more local scans could produce more actionable results. This project focuses on Ireland-wide scans, but it could be used for other similarly sized scans. The hope is that by using local knowledge and building relationships with key stakeholders (e.g. network operators, hosting providers, etc.), identified issues can be examined more closely and mitigations can be devised and implemented more effectively.

1.3 Report Outline

This report outlines how the Elastic Stack [2] is used to achieve the goals outlined previously. Elasticsearch is used for data storage and Kibana for data visualisation and analysis. The report aims to explain the process of installing and configuring the Elastic Stack, inserting the raw scan results (stored in JSON format) into Elasticsearch and visualising the results in Kibana. It outlines a command line tool developed to simplify this process. Finally, it evaluates the success of the project with respect to the above goals and outlines possible future work.

Chapter 2 outlines the background of the project and related work. It explains what internet scanning is, some of the reasons for it and some of the technologies used.

Chapter 3 explains the implementation of Elasticsearch as a data store and some of the challenges associated with it.

Chapter 4 describes the use of Kibana as a data visualisation tool and looks at some high-level visualisations and the insights provided by them.

Chapter 5 attempts to determine the success of the project by comparing implementations of the same experiment with and without the infrastructure implemented in this project both in terms of code (quantity and readability) and execution time. It also highlights some of the advantages and disadvantages of the new solution.

Chapter 6 provides some final remarks on the project.

Chapter 7 highlights some work that could be done following on from this project.

2 Background & Related Work

2.1 Internet Scanning

Internet scanning is the process of conducting network scans on an internet-wide scale.

[3] groups scans into four broad categories:

- **Vertical Scan:** a scan by a single host of multiple ports on another host.
- **Horizontal Scan:** a scan by a single host of a single port on multiple other hosts.
- **Coordinated/Distributed Scan:** horizontal scans by multiple hosts on other hosts in a /24 subnet within a one hour window.
- **Stealth Scan:** horizontal or vertical scan with low frequency with the intention of avoiding detection.

As with port and network scanning, internet scanning can be used for both good and bad. It is used by system administrators, researchers and commercial entities find new vulnerabilities, monitor the roll out of mitigations, uncovering unadvertised services such as Tor bridges and high-speed vulnerability scanning. It is also used by attackers for finding new vulnerabilities and finding targets with known vulnerabilities.

2.1.1 ZMap & ZGrab

ZMap is a fast, horizontal network scanner designed for internet scanning. It uses syn scans for port scanning and can scan the entire IPv4 address space (4,294,967,296 addresses) in under 45 minutes with a 10gigE connection [4].

ZGrab is an application layer scanner designed to work with ZMap. It is a banner grabber implemented in Go which can be used to send application-layer requests such as SSH, SMTP, POP3, IMAP and HTTPS [5]. It logs all information about each request in JSON format. More details about the data logged are given in Section 2.5. It is this JSON data that this project aims to store and analyse.

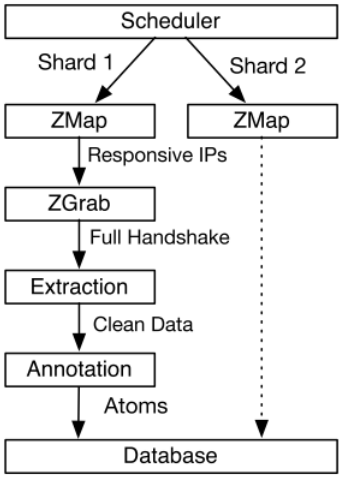

Figure 2.1: Censys.io scanning and annotation process [1]

2.1.2 Censys.io

Censys is a search engine for internet-wide scans, developed by the team behind ZMap and ZGrab [1]. It uses ZMap and ZGrab to continually scan the internet and get banner information, processes the data and stores it for searching.

ZMap is used for host discovery. The output of this seeds ZGrab, the application scanner, which produces structured JSON data containing details of the raw handshake. Another tool, ZTag, is then used to transform this data to a format which is easier to work with and add annotations with additional metadata such as device model. This metadata is not available directly from the scan results but can be derived using logic (e.g. regular expressions on protocol headers) [1]. This scan and annotation process is illustrated in Figure 2.1.

The initial version of Censys, as described in [1], used ZDb for data storage. ZDb was developed as a wrapper for RocksDB with optimisations to avoid disk access when there are no changes to data. MongoDB and Apache Cassandra NoSQL databases were initially tested, but were found to be too slow to keep up with the fast ZMap and ZGrab scans. The GitHub repository for ZDb [6] has since been marked as deprecated and archived. It is not clear what data store Censys now uses. Google Cloud Datastore is used to store historical data, allowing users to view a history for each host.

Censys provides multiple interfaces for interacting with the data. A web front-end allows users to perform full-text search and structured queries of the data. This is powered by Elasticsearch, whose indexes are updated in real time by the database. A REST API allows programmatic access to the data and supports the same functionality as the web front-end. Data is returned in JSON format. When searching using the front-end or API, the search is performed against the latest snapshot. Google BigQuery allows authenticated researchers to

perform more complex SQL queries over the full set of data, including historical data. The system architecture diagram in Figure 2.2 shows the relationship between these interfaces and the data flow.

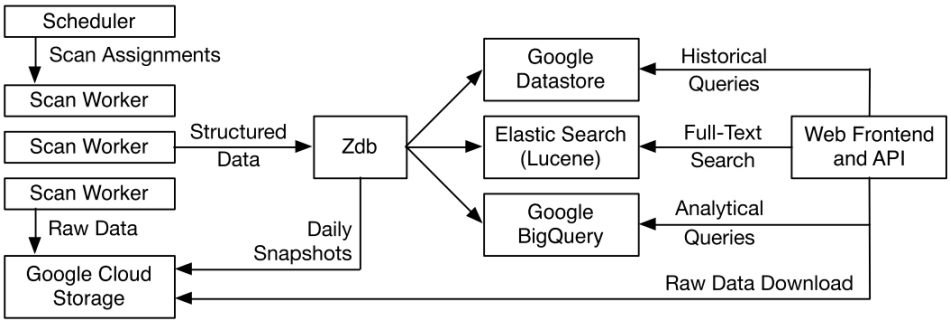

Figure 2.2: Censys.io system architecture diagram [1]

2.2 Network Scanning

Network scanning is the process of discovering hosts on a network and finding out information about them. Once the hosts on a network have been detected, port scanning can be used to identify ports open on them. Open ports show the services listening on the host, indicating the purpose of the host. For example, a host listening on port 443 is likely a web server using HTTPS. This can be used to get further details about the host by sending a HTTP request using an application-layer scanner.

2.3 Port Scanning

Port scanning involves sending client requests to ports on a host to find open ports. If a port is open, a response will be received from the host. While port scanning alone provides limited details about a host, when combined with other tools such as banner grabbers it can become very useful.

Port scanning is conducted for different purposes, some good and some bad. System administrators use port scanning for testing security policies and mapping out networks. Attackers, on the other hand, use it to find vulnerabilities and potential targets.

There are different types of ports scans [7], some of the most common being:

- **Syn Scan:** The port scanner generates and sends a TCP syn packet. If the port is open, the host will respond with a syn-ack packet. If it is closed but unfiltered, the host will respond with a reset. If no response is received, the port is likely blocked by a

firewall. As the TCP handshake is never completed, there is usually no log of the scan recorded on the host.

- **TCP Scan:** The port scanner initiates and completes the full TCP handshake. If the port is open, the handshake will complete successfully. If it does not, the port is likely closed or blocked by a firewall.

2.4 Project Background

This project builds on previous work completed on surveying cryptographic public key re-use which found that key re-use is widespread [8]. That work involved the development of a large number of bash and python scripts to get sets of IP addresses to scan, perform the scans and to analyse the results. As well as data from fresh scans, results from Censys [1] were also used to give a more comprehensive set for analysis.

IP Addresses are selected from either a previous scan or from a new set generated using a geo-location database. When using the latter, the open source MaxMind geo-location database [9] is used to get a set of addresses in a specified region. ZMap is then used to find the subset of these addresses which listen on port 25 (mail servers) to be used for further scanning. Mail servers are used as it is unlikely they are operated by or for individuals, avoiding the associated ethical issues of scanning individual people.

ZGrab is then used to connect to these addresses on a range of ports and to record the details, as outlined in section 2.5. The result of scanning an address is a JSON object which contains details about the host and each port. This object is appended to a results file. This step also verifies that each address is correctly geo-located before performing the scan. This step can take hours to days to complete, depending on the number of addresses and infrastructure used.

The analysis in this work focuses on detecting key re-use using SHA256 public key fingerprints. It uses python scripts to iterate over the scan result file, producing intermediate result files to find clusters of key re-use. JSON files and graphs are generated for each cluster. This process is relatively slow and memory intensive.

The goal of this project, as outlined in Section 1.1, is to implement a data store and data visualisation solution for use with the work described above. The hope is that this will improve efficiency and allow more general data analysis be performed more easily.

Scan Date	Source	Number of Hosts
30 November 2017	Censys.io	23,616
16 March 2018	Fresh ZGrab	24,774
15 January 2019	Fresh ZGrab	27,044

Table 2.1: Scan data used for development

2.5 Data Overview

*** NOTES ***
Scans only include data from hosts in a specific country which listen on port 25 (mail servers). Data on other ports (e.g. P443) are included when the same host is used for multiple services (e.g. mail and web server), i.e. is not a dedicated mail server

November 2017 scan data different structure - Censys data - ZTag vs raw ZGrab

2.6 Elastic Stack

Elasticsearch is a search engine and NoSQL database which supports schema-less JSON documents, based on Apache Lucene. It is highly scalable and can handle petabytes of data across hundreds of nodes [10]. It uses an open-core model, where the core functionality is open-source and additional premium features are available through subscriptions. This project utilises only the open-source features.

Elasticsearch uses a REST API for almost all interactions: managing clusters, nodes and indices and performing data CRUD and search operations.

Below is a short explanation of some basic concepts in Elasticsearch:

- **Document:** a document is the basic entity that can be indexed in Elasticsearch. It is a JSON object that contains fields which describe a single entity. In this project, a document contains the results of a single scan of a single server. Every document is associated with one index. Every document has a unique ID which is automatically generated by Elasticsearch when it is first created.
- **Index:** an index is a collection of similar documents. Every index has a unique name that is used when managing the index and when performing operations on the documents within in. For this project, each scan and scan subset has its own index. For production, it is recommended that multiple similar scans use a single index, as explained at the end of Section 3.4.1

- **Shard:** documents in an index are stored in shards. An index has one or more primary shards and every document is stored in a single primary shard. Shards allow indices to be split across multiple discs and nodes for horizontal scaling. Each shard can be searched independently, thus having multiple shards improves search performance. An index has zero or more replica shards. Replica shards are used to provide redundancy, allowing shards to be replicated across nodes. Elasticsearch manages replication and automatically switches to replica shards (if available) when the node of the primary shard fails.
- **Node:** a node is a server that runs Elasticsearch and is part of a cluster. Every node has a unique name to identify it.
- **Cluster:** a cluster is a collection of one or more nodes. Every cluster has a unique name to identify it. Nodes join clusters using the cluster name. Clusters allow horizontal scaling for performance and redundancy.

Kibana is a data visualisation tool built by the Elastic team that integrates with Elasticsearch.

2.7 Transmission Control Protocol

Transmission Control Protocol (TCP) is a connection-oriented, reliable transport layer protocol designed to be used between two hosts which are on the same network or are on interconnected networks [11, 12]. It provides error detection, re-transmission, sequencing and flow control. TCP has three main phases of operation: connection establishment (TCP Handshake), data communication and connection closing [12].

2.8 Application Layer Internet Protocols

2.8.1 SSH

The Secure Shell (SSH) protocol is a network protocol which allows secure remote access over insecure networks [13]. It provides an encrypted tunnel between the client and the server, as well as client and server authentication. It runs over TCP and has been assigned port 22 by IANA [14].

2.8.2 SMTP

Simple Mail Transfer Protocol (SMTP) is an electronic mail protocol designed to transfer mail reliably and efficiently. SMTP can run over any transport service which provides a reliable ordered data stream, most commonly TCP [15]. IANA has assigned ports 25 and 587 to SMTP [14]. Port 25 is most commonly used between mail servers, while port 587 is reserved for email message submission [16].

2.8.3 POP3

Post Office Protocol version 3 (POP3) is an electronic mail retrieval protocol used by email clients to retrieve mail that a mail server is holding for it[17]. Once an email has been retrieved by the client, it is deleted from the server. It runs over TCP and has been assigned port 110 by IANA [14].

2.8.4 IMAP & IMAPS

Internet Message Access Protocol (IMAP) is an electron mail protocol which allows a client to read manipulate emails on a server [18]. Unlike POP3, emails are not deleted on the server after they are read by a client. This allows synchronisation of a single mailbox across multiple clients, among other advantages. It can run on any reliable data stream, usually TCP. It has been assigned port 143 by IANA [14].

IMAPS is a secure version of IMAP which uses TLS. While IMAPS can be used on port 143 using the STARTTLS mechanism, it is recommended that port 993 is instead used for implicit TLS [14, 19].

2.8.5 HTTPS

Hypertext Transfer Protocol Secure (HTTPS) is a secure version of HTTP which uses TLS. It can be run over any transport service which provides a reliable connection-oriented data stream, usually TCP [20]. It has been assigned port 443 by IANA [14].

2.9 Transport Layer Security

Transport Layer Security (TLS) is a cryptographic protocol which provides secure communication over a computer network. Symmetric cryptography is used to encrypt data

between two applications using keys unique to that connection, ensuring a private connection. The keys are based on a shared secret, negotiated during the TLS Handshake. Message authentication codes, computed using secure hash functions, are used to ensure a reliable connection [21]. The TLS Handshake protocol is used to set up the secure connection.

2.9.1 TLS Handshake

The TLS Handshake is initiated by the client once the TCP connection has been established.

1. **Client Hello:** the client sends a hello message to the server. In it, the client states the version of TLS and the cipher suites it supports, as well as a client nonce (random number).
2. **Server Hello:** the server chooses the highest mutually supported TLS version to use. It chooses a cipher suite from the list of those supported by the client and generates a server nonce (random number). It then sends these three things along with its TLS certificate.
3. **Server Verification:** the client attempts to verify the server's TLS certificate with the issuing Certificate Authority to ensure the server is who it says it is.
4. **Pre-Master Secret:** the client generates another random number called the Pre-Master Secret (PMS). It gets the server's public key from the certificate, encrypts the PMS using the key and sends the encrypted PMS to the server.
5. **Master Secret:** using the client nonce, server nonce and PMS, the client and server independently compute the master secret using the same key derivation function, as specified by the TLS standard [21]. They should both get the same result. The master secret is then split up to give a client MAC key, a server MAC key, a client encryption key and a server encryption key.
6. **Client Ready:** the client sends a message, encrypted using the derived client encryption key, to the server to indicate that it is ready to use encrypted messages from now on.
7. **Server Ready:** the server decrypts and verifies the message received from the client and sends a message to the client indicating that all further messages will be encrypted.

2.9.2 Cipher Suite

A cipher suite is a set of algorithms that are used to secure a connection in TLS. A set contains a key exchange algorithm, a symmetric encryption algorithm and a message authentication code (MAC) algorithm. When RSA is used for key exchange, it can also be used to authenticate the server - if the server successfully decrypts the pre-master secret encrypted by the client using the servers public key, it demonstrates that it knows the private key. When Diffie-Hellman is used for key exchange, either fixed parameters can be provided by the server, or it can send temporary parameters using the server key exchange message.

2.9.3 Public Key Fingerprint

A public key fingerprint is a cryptographic hash of a public key. As it is shorter than the key it represents, it can be used in place of the key to simplify some tasks. This project focuses on using the SHA256 hash of public keys.

2.9.4 SHA256

A hash function takes an input x and computes a string of fixed size $H(x)$. A cryptographic hash function is one for which it is computationally infeasible to find two messages x and y which give $H(x) = H(y)$ [22] and given $H(x)$ it should be infeasible to find x. In other words, it should be infeasible to (1) find two messages which produce the same hash and (2) find a message that corresponds to a given hash [23]. SHA256 is a Secure Hash Algorithm whose output is 256 bits in length.

3 Data Store

This chapter outlines the configuration of Elasticsearch for this project, explains some of the features used and outlines challenges encountered. It also describes a command line tool developed as part of this project to simplify the process of installing and managing Elasticsearch and working with the data.

3.1 Data Types and Mappings

Elasticsearch uses mappings to define how to store and index documents and fields. A mapping can be used to define the type and the format of a field where applicable (e.g. a date). Dynamic mapping allows documents to be inserted without explicitly defining mappings for every field. This was essential for this project for a number of reasons:

- **Number of Fields:** there were over 5,000 unique fields in the data from the three scans used for this project. Explicitly defining mappings for each of these would be difficult and tedious.
- **Extensibility:** future work may extend scanning to additional protocols or add additional data processing and transformations. Dynamic mapping allows these changes to be made without the need to make any changes to Elasticsearch.

There were cases, however, where explicit mappings were needed as the correct types or formats were not detected automatically.

- **IP:** Elasticsearch has an IP datatype which supports IPv4 and IPv6 addresses. It allows searching IP addresses within ranges and using CIDR notation. When using dynamic mapping, the IP field was detected to be of type string.
- **Geo-Point:** the geo-point data type stores latitude and longitude pairs which can be used to plot points on a map, as shown in Section 4.1.2. Without an explicit mapping, the latitude and longitude fields were represented as independent number fields.
- **Date:** explicitly defining the format for the date fields ensures the date is returned in

the expected format in search results.

3.2 Index Templates

Elasticsearch allows index configuration settings to be set when creating the index, using the *_settings* API endpoint or in an index template. An index template is a JSON file which contains settings and mappings to be used when creating matching templates. Table 3.1 shows the settings used in the default index template in this project. The template also contains the mappings explained in Section 3.1.

The *index_patterns* field is used to set indices the template should be applied to. The template will be applied to an index if the index name matches the value pattern. The field takes an array which can be used to give multiple index names to which the template will be applied. Wildcards can be used in the patterns.

Multiple templates can match an index name.The *order* field is used to determine the order in which templates are applied. Templates are applied in order from lowest to highest. Higher order templates take precedence over lower order ones. This allows general settings and mappings to be set for all indices and more specific ones to be set for certain indices.

The *index.mapping.total_fields.limit* field is used to define the total number of unique fields an index can contain. Information about each field is stored in memory, so having too many fields can cause memory errors. Testing for this project showed an average of 3,000 fields per index when separate indices were used for each scan. If data from different sources (e.g. Censys.io and fresh ZGrab) were stored in the same index, this field would need to be increased. There are 5,610 unique fields when the data from all three scans are combined. The number of unique fields should not vary significantly between similar scans. If the number was significantly different, it may indicate an error, e.g. encoding issues or other problems with the raw JSON. If the structure of the JSON changed significantly, e.g. as a result of new data transformations, the data should be added to a new index.

The *number_of_shards* field is used to set the number of primary shards for an index. This can only be set when an index is created; it can't be changed later using the *_settings* API.

The *number_of_replicas* field is used to set the number of replica shards for an index. Having a replica shard on the same node is of little benefit. Replica shards on separate nodes provide a backup in the case of failure of a node. The maximum number of replica shards should be $(\#nodes - 1)$. In this case, one node will host the primary shard and every other node will host a replica of it.

The *version* field is optional and is not used by Elasticsearch internally. It is intended to simplify template management by administrators or other systems. The *_template* API endpoint can be used to get version of a template used when creating an index.

Changes to index templates are only applied to new indices created after the template is update and not to existing templates.

Field Name	Value	Explanation
index_patterns	["*"]	This template will be applied to all new indices.
order	1	Using order 1 overrides Elasticsearch default templates (order 0).
index.mapping.total_fields.limit	5000	Allows true fields to be set but prevents mapping explosion in the case of an error.
number_of_shards	1	Section 3.4.1 explains why 1 primary shard is used for this project.
number_of_replicas	0	Single node cluster with no replication.
version	1	

Table 3.1: Settings in default Elasticsearch index template used for this project.

3.3 CLI Tool

3.3.1 Installing Elasticsearch & Kibana

```
Usage: esman installdeps [OPTIONS]

  Install Elasticsearch and Kibana, enable auto-start and create
  Elasticsearch templates

Options:
  -h, --help  Show this message and exit.
```

Figure 3.1: Command to install Elasticsearch and Kibana

Figure 3.1 shows how to install the Elastic Stack components used for this project. It installs Java, Elasticsearch and Kibana, enables the services to run them automatically on startup and creates the default Elasticsearch templates.

3.3.2 Extracting Subset

```
Usage: esman extractsubset [OPTIONS]

  Read a CSV file, one (default second) column of which is a URL, then
  extract matching records from a JSON file

Options:
  -c, --csvinfile TEXT    CSV file containing list of domains  [required]
  -j, --jsoninfile TEXT   JSON file containing list of scan records
                          [required]
  -o, --output_file TEXT  JSON file in which to put records (one per line)
                          [required]
  --col INTEGER           Column from input file that has the URL (default =
                          1)
  --tryip                 If set, searches json_infile for IP if no records
                          match domain
  --debug                 If set, turns on verbose logging
  -h, --help              Show this message and exit.
```

Figure 3.2: Command to extract data subset

Figure 3.2 shows how to extract a subset of data from a JSON file. This is intended to be used with the *AddDNSDetail* script developed previously as part of the overall project [24], but it can also be used independently. It takes a CSV file which contains a list of domain names and optionally a list of IPs and a JSON file to search. For each domain in the CSV file the entire JSON input file is searched record by record. If a match for the domain is found in a record, the record is added to the output fie. If no match is found for the domain and the *tryip* flag is set, the input file is again searched, this time using the IP.

This is an imperfect solution, but it is sufficient for this project. As it searches entire records, if one domain is referenced by another, it will match for both domains. For example in the aviation subset used for this project, the server for Ireland West Airport responded on port 443. On the homepage, there are references to the Ryanair and Aer Lingus websites. As a result, the record for Ireland West Airport matched for ryanair.com, aerlingus.com and irelandwestairport.com.

3.3.3 Inserting Data

```
Usage: es insert [OPTIONS]

  Insert JSON Data to Elasticsearch

Options:
  -i, --input TEXT              JSON File to be inserted  [required]
  --index TEXT                  Elasticsearch index to be inserted to
                                [required]
  -rd, --rundate <YYYY-MM-DD>   Run date to be used for Kibana
  -cc, --countrycode TEXT       Two letter country code to be used for Kibana
  -h, --help                    Show this message and exit.
```

Figure 3.3: Command to insert JSON file into Elasticsearch

The raw data for this project is the JSON output from ZGrab, as described in Section 2.5. Each JSON file contains approximately 25,000 records and is 800MB - 1GB in size. With 2GB RAM available and Elasticsearch needing at least 1GB, it is not feasible to load and insert the entire file in one go. To overcome this, a python generator is used with the Elasticsearch python library to load a fixed number of records at a time and insert them into Elasticsearch in bulk. Before sending the data to Elasticsearch, geolocation information is added for each record using code developed in the original work [25]. Each scan file took approximately four hours to insert. A progress bar and the estimated time to completion are displayed to the user. Figure 3.3 shows the command to insert data and the options.

3.3.4 Managing Indices

```
Usage: es viewallindices [OPTIONS]

  View all Elasticsearch indices and their status

Options:
  -h, --help  Show this message and exit.
```

Figure 3.4: Command to view all indices

Figure 3.4 shows the command to view all Elasticsearch indices and information about them including number of primary and replica shards, number of documents and health status. This was used frequently when developing the code to insert the data in bulk and overcoming the problem described in Section 3.4.1. It is much more convenient than using the alternative command shown in Listing 1.

```
\$ curl -X GET "localhost:9200/_cat/indices?v"
```

Listing 1: Curl command to view indices

```
Usage: es deleteindex [OPTIONS]

  Delete an Elasticsearch index. WARNING: Cannot be undone

Options:
  -i, --index TEXT  Name of the index to delete  [required]
  --yes             Confirm the action without prompting.
  -h, --help        Show this message and exit.
```

Figure 3.5: Command to delete index

Figure 3.5 shows the command to delete an Elasticsearch index. As with the previous command to view indices, this was added for convenience to replace the cumbersome alternative shown in Listing 2. It also adds a layer of safety by using a confirmation prompt. This can be skipped using the *–yes* flag to allow use in scripts.

```
\$ curl -X DELETE "localhost:9200/indexname"
```

Listing 2: Curl command to delete an index

3.4 Challenges

3.4.1 Elasticsearch Crash on Bulk Insert

The biggest challenge and one of the most time consuming parts of this project was successfully inserting the raw data into Elasticsearch in bulk. It was initially anticipated that this would be a straightforward task - read the data from the JSON file and insert it into Elasticsearch. The first attempt at doing this used a Python script to read 100 JSON records from the raw data file and send them to Elasticsearch in a bulk insert request.

The output from the python script showed that the data was being read from the file and sent to Elasticsearch. However, checking the number of documents in the Elasticsearch index showed that a very small number of records was being stored by Elasticsearch compared to the number of records sent to it. Initially, the requests were being queued by Elasticsearch and could be seen in Elasticsearch's thread pool. After approximately 200-300 requests, Elasticsearch would close all active HTTP connections and refuse any further incoming insert requests. The python script would log the HTTP connection timeout and

attempt to re-establish a connection. Again viewing the number of documents in the index would show few-to-no additional records had been inserted. The thread pool showed that the requests were being removed from the queue and marked as rejected. At this stage, Elasticsearch would become sluggish and slow to respond to status requests. Eventually, Elasticsearch would crash.

Several attempts were made to overcome the problem. The chunk size (number of records per request) was reduced first to 50 and then to 10, and the max size of the request in bytes was reduced from the default of 100MB to 5MB. Sleep timers were added to the python script to control the rate at which insert requests were sent. After a record was read from the JSON file, the script would wait for one second before continuing. After a request was sent, the script would wait for ten seconds before starting the next read. This meant an insert request was being sent approximately every 60 (10 + 50) or 20 (10 + 10) seconds, depending on the chunk size. As the python script and the Elasticsearch instance were running on the same machine, a check was also added to ensure Elasticsearch was running before a request was sent. If Elasticsearch wasn't running (i.e. had crashed), the Elasticsearch service would be started again and the script would wait for 20 seconds before sending the request. While initial indications appeared to suggest these changes were successful, the process completed after approximately 10 hours with only 18,000 / 24,700 records stored.

*** NOTES ***
ES & gc logs - gc active more than code
ES deep-dive
Elasticsearch organises documents into indices. Each index consists of one or more shards. A shard is associated with a single Lucene index. A document is stored in one primary shard and zero or more replica shards. As each primary shard is independent of the others, multiple shards can be searched in parallel and the results can be combined. Replica shards can be used to provide redundancy by replicating the data across nodes. Figure 3.6 shows an Elasticsearch cluster with two nodes. Each index has three shards, each with one replica.

Data in shards are stored in immutable Lucene segments. The number of segments varies over time as segments are periodically merged into larger segments. Segments are immutable, so the new segment must be created before the old ones can be deleted. This leads to a fluctuation in disk space. Merging is a resource intensive process.

Elasticsearch stores cluster state in memory. This contains information about mappings for every field in every index, state information about every shard and information about where data is physically stored on disk for every segment in each shard [26]. This means overhead is directly proportional to the number of fields, indices and shards. Elasticsearch recommend

Figure 3.6: Elasticsearch architecture with one cluster, two nodes, two indexes with three shards, each with one replica

a maximum of 20 shards per GB heap available and a heap size of 50% of available memory. By default, new indices created in Elasticsearch have five primary shards, each with one replica, giving ten shards per index.

This project used a single machine with 2GB of RAM and 6GB of swap memory available. Using the default configuration and following the recommended limits, this would allow a maximum of two indices without Kibana. Kibana uses an index with a single primary shard to store objects, as well as two single-shard indices per day for the current day and previous seven days containing monitoring information about Elasticsearch and Kibana. This results in 17 system indices and shards. Creating a single index with the default configuration alongside the system indices exceeds the recommended limits.

For development and testing, a new index was created for each scan and scan subset. Each of these indices had over 5,400 unique fields. While the fields were different for the 2017 data, they were the same for 2018 and 2019. As they were created as separate indices, however, Elasticsearch treats them as separate fields, effectively storing information about them twice in the cluster state.

To overcome these issues, a number of changes were implemented.

- **Heap Size:** the heap size was set to 1GB, 50% of the RAM available.
- **Index Templates:** index templates were created which allowed the number of primary and replica shards to be defined. As the average number of documents was

relatively small (25,000 records per scan), the heap size was limited and search performance was not a priority, the number of primary indices was set to 1. With only a single node cluster for development, the number of replica shards was set to 0.

- **Number of Indices:** frequent changes in mapping types and fields made it more convenient to use separate indices for each data set during development. For production use, however, a different scheme would be required. Depending on the frequency of scans, a single index could be used for a month of daily scans or a year or more of monthly scans of a specific country or region. This would reduce the amount of duplicate field information in cluster state, reducing overhead.

4 Data Analysis

4.1 Kibana

4.1.1 Web Server and TLS Certificate

4.1.2 Visualisations

*** NOTES ***
Creating a visualisation: The question asked was 'of the servers who share the same fingerprint, what cipher suites do they use?'. If just one server uses a weak cipher suite, it leaves the rest of the servers more vulnerable; this information could be shared with those vulnerable servers. If the servers are owned/run by two separate entities, those with the strong cipher suite(s) would be unaware of the vulnerability and believe that by using a strong cipher suite, they are safe in that regard.

Kibana bar charts require two basic things - a metric (aggregation) for the y-axis and a bucket (grouping) for the x-axis. In this case, we want to count the number servers who have the same sha256 fingerprint on port 443, so our metric for the y-axis is simply 'Count'.

We want to group the servers by sha256 fingerprint and display the fingerprints along the x-axis in decreasing order by count (i.e. display the most used fingerprint first, followed by the second most used, etc.). Therefore, we set our primary x-axis bucket to be the p443 sha256 fingerprint field and set 'order by' to count. Note that, because we have not yet subdivided our data any further, the count used for ordering is the total number of records in the data which contain a given key, regardless of run. This means if a server is present in two scans, it will be counted twice for the purpose of ordering.

To overcome this in our graph, we create a sub-bucket using the date histogram aggregation to split the chart by run date. This gives us accurate counts for each fingerprint and allows us to view the change in usage over time by showing each run as a separate sub-graph.

Finally, within each run, and for each fingerprint, we want to breakdown the count by the

cipher suite used. To do this, we create a sub-bucket using the terms aggregation with the p443 cipher suite field.

Below is the Elasticsearch query to do this. The aggregation fields are as explained above. Size 0 tells Elasticsearch to only return the aggregation results, not the source documents. We don't want to filter the data in any way, so we use "match_all " to include all documents.

4.2 Dashboards

4.2.1 Ireland Data

4.2.2 Aviation Data Subset

4.3 Results/Insights

5 Evaluation

5.1 Same Keys

5.1.1 Lines of Code

5.1.2 Run Time

5.2 Scalability

5.3 Usability

6 Conclusion

7 Future Work

Bibliography

[1] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, and J. Alex Halderman. A Search Engine Backed by Internet-Wide Scanning. *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security - CCS '15*, pages 542–553, 2015. ISSN 15437221. doi: 10.1145/2810103.2813703. URL http://dl.acm.org/citation.cfm?doid=2810103.2813703.

[2] Elastic. Elastic Stack. https://www.elastic.co/, 2019. [Online; accessed 11-April-2019].

[3] Vinod Yegneswaran, Paul Barford, and Johannes Ullrich. Internet intrusions: Global characteristics and prevalence. *ACM SIGMETRICS Performance Evaluation Review*, 31 (1):138–147, 2003.

[4] Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. Zmap: Fast internet-wide scanning and its security applications. In *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*, pages 605–620, Washington, D.C., 2013. USENIX. ISBN 978-1-931971-03-4. URL https://www.usenix.org/conference/usenixsecurity13/technical-sessions/paper/durumeric.

[5] ZMap Team. The ZMap Project. https://zmap.io/, 2019. [Online; accessed 11-April-2019].

[6] ZMap Team. ZDb. https://github.com/zmap/zdb, 2019. [Online; accessed 11-April-2019].

[7] Stuart Staniford, James A Hoagland, and Joseph M McAlerney. Practical automated detection of stealthy portscans. *Journal of Computer Security*, 10(1-2):105–136, 2002.

[8] Stephen Farrell. Clusters of re-used keys. Cryptology ePrint Archive, Report 2018/299, 2018. https://eprint.iacr.org/2018/299.

[9] Max Mind. Open Source Geo-Location Database. https://www.maxmind.com/en/open-source-data-and-api-for-ip-geolocation, 2019. [Online; accessed 17-April-2019].

[10] Elastic. Elasticsearch. https://www.elastic.co/products/elasticsearch, 2019. [Online; accessed 15-April-2019].

[11] Vinton Cerf and Robert Kahn. A protocol for packet network intercommunication. *IEEE Transactions on communications*, 22(5):637–648, 1974.

[12] J. Postel. Internet Protocol. RFC 791 (Internet Standard), September 1981. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc791.txt. Updated by RFCs 1349, 2474, 6864.

[13] T. Ylonen and C. Lonvick (Ed.). The Secure Shell (SSH) Protocol Architecture. RFC 4251 (Proposed Standard), January 2006. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc4251.txt. Updated by RFC 8308.

[14] M. Cotton, L. Eggert, J. Touch, M. Westerlund, and S. Cheshire. Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry. RFC 6335 (Best Current Practice), August 2011. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc6335.txt.

[15] J. Postel. Simple Mail Transfer Protocol. RFC 821 (Internet Standard), August 1982. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc821.txt. Obsoleted by RFC 2821.

[16] R. Gellens and J. Klensin. Message Submission for Mail. RFC 6409 (Internet Standard), November 2011. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc6409.txt. Updated by RFC 8314.

[17] J. Myers and M. Rose. Post Office Protocol - Version 3. RFC 1939 (Internet Standard), May 1996. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc1939.txt. Updated by RFCs 1957, 2449, 6186, 8314.

[18] M. Crispin. INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1. RFC 3501 (Proposed Standard), March 2003. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc3501.txt. Updated by RFCs 4466, 4469, 4551, 5032, 5182, 5738, 6186, 6858, 7817, 8314, 8437, 8474.

[19] K. Moore and C. Newman. Cleartext Considered Obsolete: Use of Transport Layer Security (TLS) for Email Submission and Access. RFC 8314 (Proposed Standard), January 2018. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc8314.txt.

[20] E. Rescorla. HTTP Over TLS. RFC 2818 (Informational), May 2000. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc2818.txt. Updated by RFCs 5785, 7230.

[21] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc5246.txt. Obsoleted by RFC 8446, updated by RFCs 5746, 5878, 6176, 7465, 7507, 7568, 7627, 7685, 7905, 7919, 8447.

[22] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach (6th Edition)*. Pearson, 6th edition, 2012. ISBN 0132856204, 9780132856201.

[23] D. Eastlake 3rd and T. Hansen. US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF). RFC 6234 (Informational), May 2011. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc6234.txt.

[24] Stephen Farrell. Internet Scanning. https://github.com/sftcd/scans, 2018. [Online; accessed 15-April-2019].

[25] Stephen Farrell. Internet Surveys. https://github.com/sftcd/surveys, 2018. [Online; accessed 15-April-2019].

[26] Elastic. How many shards should I have in my Elasticsearch cluster? https://www.elastic.co/blog/how-many-shards-should-i-have-in-my-elasticsearch-cluster, 2017. [Online; accessed 14-April-2019].

A1 Appendix