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

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

List of Figures

List of Tables

1 Introduction

1.1 Goal

The goal of this project is to design and implement a data store and data visualisation tool for a localised internet scanning infrastructure which supports Ireland-sized internet scans. Specifically, this report focuses on outlining how to use the Elastic Stack (2) for this purpose. The project uses Elasticsearch for data storage and Kibana for data visualisation and analysis. These should be able to run on a single machine with limited resources. 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.

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 Structure

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 TCP

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 (3, 4). 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 (4).

2.2 Application Layer Internet Protocols

2.2.1 SSH

The Secure Shell (SSH) protocol is a network protocol which allows secure remote access over insecure networks (5). 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 (6).

2.2.2 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 (7). IANA has assigned ports 25 and 587 to SMTP (6). Port 25 is most commonly used between mail servers, while port 587 is reserved for email message submission (8).

2.2.3 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(9). 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 (6).

2.2.4 IMAP & IMAPS

Internet Message Access Protocol (IMAP) is an electron mail protocol which allows a client to read manipulate emails on a server (10). 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 (6).

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 (6, 11).

2.2.5 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 (12). It has been assigned port 443 by IANA (6).

2.3 Transport Layer Security (TLS)

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 (13). The TLS Handshake protocol is used to set up the secure connection.

2.3.1 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 (13). 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.

2.3.2 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.

2.3.3 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.

2.4 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 (14), 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.

2.5 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.

2.6 Internet Scanning

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

(15) 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.

2.6.1 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 (16).

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 (17). It logs all information about each request in JSON format. More details about the data logged are given in Section 2.7. It is this JSON data that this project aims to store and analyse.

2.6.2 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,

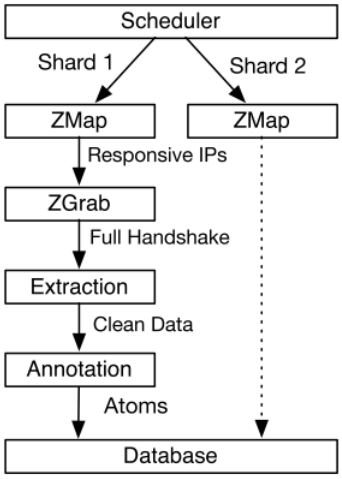

Figure 2.1: Censys.io scanning and annotation process (1)

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 (18) 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.

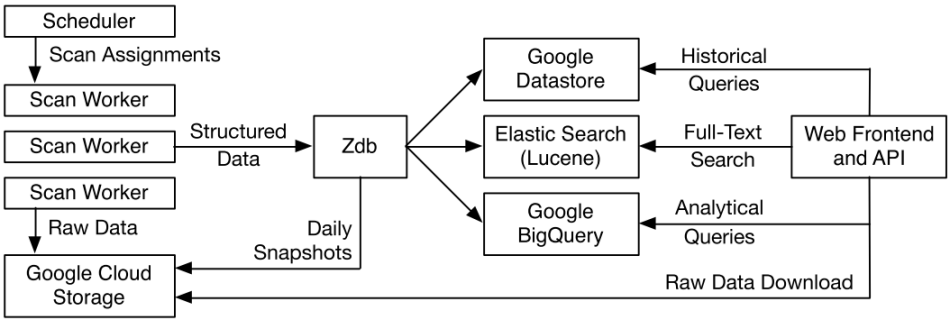

Figure 2.2: Censys.io system architecture diagram (1)

2.7 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

3 Data Store

3.1 Elasticsearch

3.2 Inserting Data

3.3 Data Types and Mappings

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 3.1 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 (19). This means overhead is directly proportional to the number of fields, indices and shards. Elasticsearch recommend 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

Figure 3.1: Elasticsearch architecture with one cluster, two nodes, two indexes with three shards, each with one replica

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. 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/`, . [Online; accessed 11-April-2019].

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

[4] Jon Postel. Internet protocol. Technical report, 1981.

[5] Tatu Ylonen and Chris Lonvick. The secure shell (ssh) protocol architecture. Technical report, 2005.

[6] Michelle Cotton, Lars Eggert, Joe Touch, Magnus Westerlund, and Stuart Cheshire. Internet assigned numbers authority (iana) procedures for the management of the service name and transport protocol port number registry. Technical report, 2011.

[7] Jon Postel. Simple mail transfer protocol. Technical report, 1982.

[8] Randall Gellens and J Klensin. Message submission for mail. Technical report, 2011.

[9] John Myers and Marshal Rose. Post office protocol-version 3. Technical report, 1996.

[10] Mark Crispin. Internet message access protocol-version 4rev1. Technical report, 2003.

[11] K Moore and C Newman. Cleartext considered obsolete: Use of transport layer security (tls) for email submission and access. Technical report, 2018.

[12] Eric Rescorla. Http over tls. Technical report, 2000.

[13] Tim Dierks and Eric Rescorla. The transport layer security (tls) protocol version 1.2. Technical report, 2008.

[14] 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.

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

[16] ZMap: Fast Internet-wide Scanning and Its Security Applications ZMap: Fast Internet-Wide Scanning and its Security Applications. In *Proceedings of the 22nd USENIX Security Symposium*, 2013. ISBN 978-1-931971-03-4. URL `https://zmap.io/`.

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

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

[19] 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`, . [Online; accessed 14-April-2019].

A1 Appendix