BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

VISUALIZATION SYSTEM OF NETWORK FORENSIC DATA

SYSTÉM PRO VIZUALIZACI SÍŤOVÝCH FORENZNÍCH DAT

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **IVAN MANOILOV**
AUTOR PRÁCE

SUPERVISOR **Ing. JIŘÍ HYNEK, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2021

Department of Information Systems (DIFS) Academic year 2020/2021

Bachelor's Thesis Specification

Student: **Manoilov Ivan**
Programme: Information Technology
Title: **Visualization System of Network Forensic Data**
Category: Information Systems
Assignment:

1. Get acquainted with the problem of processing and visualization of network forensic data, particularly with bidirectional flows.
2. Study existing possibilities how to visualize data on dashboards. Focus on the chart types suitable for visualization of network forensics data. Study web technologies suitable for this purpose.
3. Get acquainted with the existing solution using Grafana system. Analyze its strengths and weaknesses.
4. Design an information system composed of a set of dashboards presenting information gathered from the analysis of bidirectional flows. Respect the results of the analysis.
5. Implement the designed solution.
6. Test the solution and evaluate the achieved results.

Recommended literature:

- Ryšavý, O., and Hynek, J.: *Visual Representation of Network Forensic Data*. Brno University of Technology, Faculty of Information Systems, 2020, Technical Report, FIT-TR-2020-07.
- Few, S.: *Information Dashboard Design: The Effective Visual Communication of Data*. O'Reilly, 2006, ISBN: 978-059-6100-162.
- Jacobs, J, and Rudis B.: *Data-Driven Security: Analysis, Visualization and Dashboards.* John Wiley & Sons, 2014, ISBN: 978-111-8793-824.
- Miller, P. M.: *Visualization for network forensic analyses: extending the Forensic Log Investigator (FLI).* Thesis, Iowa State University, 2008.
- Johnson, J.: *Designing with the Mind in Mind: Simple Guide to Understanding User Interface Design Guidelines*. Morgan Kaufmann Publishers/Elsevier, 2010, ISBN: 978-0-12-375030-3.

Requirements for the first semester:

- Items 1 to 4

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor: **Hynek Jiří, Ing., Ph.D.**
Head of Department: Kolář Dušan, doc. Dr. Ing.
Beginning of work: November 1, 2020
Submission deadline: May 12, 2021
Approval date: October 27, 2020

Abstract

This bachelor thesis focuses on creating a system for the visualization of network forensic data. The concept of a dashboard and web application architecture is explained. It analyses the best useful visualizations for forensic analysis of network data. The problem of processing `.PCAP` files and data structures is addressed as well. Furthermore, data aggregation techniques are described and explained. The process of implementation of the system is described and illustrated. Finally, testing and benchmarking results are shown, and the whole application is evaluated according to them.

Abstrakt

Tato bakalářská práce se zaměřuje na vytvoření systému pro vizualizaci síťových forenzních dat. Vysvětluje koncept dashboardu a architektury webové aplikace. Analyzuje vhodné vizualizace pro forenzní analýzu síťových dat. Rovněž je řešen problém zpracování `.PCAP` souborů a datových struktur. Následně jsou popsány a vysvětleny techniky agregace dat. Je popsán a ilustrován postup implementace systému. Nakonec jsou uvedené výsledky testování a benchmarkingu a podle nich je pak aplikace vyhodnocena.

Keywords

React, JavaScript, Kotlin, Spring Framework, TimescaleDB, network forensic analysis, web application

Klíčová slova

React, JavaScript, Kotlin, Spring Framework, TimescaleDB, síťová forenzní analýza, webová aplikace

Reference

MANOILOV, Ivan. *Visualization System of Network Forensic Data*. Brno, 2021. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Jiří Hynek, Ph.D.

Rozšířený abstrakt

Práce se zaměřuje na vytvoření systému pro vizualizaci síťových forenzních dat. Cílem práce je navrhnout webovou aplikaci, která umožní uživatelům zpracovávat a analyzovat sítová data.

V první části práce je popsána samotná forenzní síťová analýza. Jsou popsané základní datové jednotky analýzy jako paket a flow. Další pozornost je věnovaná způsobu sbírání síťových dat přes tzv. *packet sniffers*. Detailně je v této části rozebrán i nástroj pro tento účel – Wireshark. Následně je analyzované rozhraní tohoto nástroje, zejména formát výstupů jeho činnosti a knihovny umožňující používat stejnou funkcionalitu. Nakonec popisuji síťovou forenzní analýzu, její postup a cíle. V rámci popisu analyzuji i čtyři případy užití forenzní analýzy pro tuto práci.

Další kapitola je věnovaná vizualizaci síťových forenzních dat a nástroji *dashboard*. Na začátku definuji dashboard, jeho typy a jejich rozdíly, uvádím ukázkové obrázky dashboardu. Následně analyzuji typy diagramů, které jsou vhodné pro síťovou forenzní analýzu. Detailní popis čtyř grafů (spojnicový, sloupcový, Sankey, chord a síťový) rovněž obsahuje případy užití daných diagramů, stejně jako ukázkové obrázky.

Následující kapitola popisuje problém, jeho analýzu, požadavky na systém a srovnaní existujících nástrojů. Kromě problému jsem detailně vysvětlil výhody použití systémů, oproti sítím, které nepoužívají žádné nástroje pro analýzu a vizualizaci. Doplněním k analýze problému je také i analýza a definice cílové skupiny uživatelů. Abych mohl porovnat existující nástroje, potřeboval jsem přesně definovat požadavky na systém a výkon. Nakonec jsem na základě těchto požadavků porovnal současné nástroje jako Wireshark, Network Miner, Snort, Splun, a vypsal jejich výhody a nevýhody.

Po upřesnění požadavků a porovnaní existujících systémů jsem mohl provést návrh svého řešení. Návrh se skládá z rozšířených požadavků na implementaci systému jako například technologické potřeby a omezení vybraných knihoven a frameworků, definice API. Následně, jsem rozšířil požadavky ze strany back-endu a přesně popsal očekávaní od jednotlivých modulů a jejich spolupráci. Minimální obsah back-endu se skládá ze dvou modulů: zpracování `PCAP` a poskytnutí dat do front-endu. Každý modul je podpořen diagramem upřesňujícím vzájemné vztahy entit. Poté se zabývám architekturou systému a popisuji databázovou vrstvu. Prostřednictvím schémat jsem vypsal přesné požadavky na implementaci. Doporučuji zde používat databázový systém umožňující optimální prací s časovanými daty. Ve finální části kapitoly jsem popsal požadavky na front-end a základní obrazovky. Jsou jimi dvě – okno pro nahrávaní `PCAP` souborů a hlavně dashboard. Uvádím i přibližný design daných obrazovek pomocí skici.

Nejrozsáhlejší kapitola popisuje implementaci systému a použité technologie, stejně jako důvody jejich použití. Systém se skládá ze tří částí: webové aplikace, webového serveru a databáze. Propojení mezi jednotlivými částmi systému je implementované pomoci `REST` API. Základ webového serveru je implementován pomocí frameworku `Spring`, který je knihovnou pro `Java Virtual Machine` jazyky. `Spring Framework` umožňuje jednoduchou práci se databází, webovými rozhraními a dalšími užitečnými částmi rozsáhlého systému jako vkládání závislostí a použití vice vláken. Následně popisuji použití programovacího jazyka webového serveru – Kotlin. Důležitou části popisu serveru jsou jeho jednotlivé řídicí prvky a zároveň základní buňky – kontroléry. Vysvětluji přesnou implementaci kontrolérů pro pasovaní `PCAP` a pro poskytnuti dat do dashboardu, dodatečné ukazuji diagramy architektury těchto kontrolérů a modulu, který řídi. Nakonec vysvětluji organizační detaily jako modulová struktura a nástroje pro sestavení systému.

Druhou částí implementace je webová aplikace. Pro tento účel jsem využil `React`, jehož výhody jsem popsal v dalších sekcích. Základním prvkem pro webovou aplikaci je i řízení stavu, které je zajištěné knihovnou `Redux`. Dodatečně popisuji jednotlivé části stavu, jejich interakce a způsoby změny. Nakonec detailně popisuji obrazovky webové aplikace pomocí snímků obrazovek.

Testovaní jsem provedl pomocí manuálních end-to-end testů, jejichž definice je označena v další kapitole. Výsledkem těchto testů je celkové hodnocení implementovaného systému. Výstupy testu byly uspokojující a ukázaly pár drobných chyb, které byly opraveny. Popisuji i větší chyby a návrh na jejich řešení. Nakonec jsem provedl testy výkonnosti dvou časové náročných částí systému: zpracovaní `PCAP` a datového shlukovaní. Výsledky jsou zobrazené v tabulkách ve stejné sekci.

Budoucí vývoj systému může byt zaměřen na automatické shlukovaní dat podle metadat `PCAP` a hlubší pokrytí všech případů užití.

Visualization System of Network Forensic Data

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the academic supervision of Ing. Jiří Hynek, Ph.D. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

. . . .
Ivan Manoilov
May 11, 2021

Acknowledgements

I would like to thank my supervisor Ing. Jiří Hynek, Ph.D. for his guidance and advice.

Contents

1 Introduction **3**

2 Processing of Network Forensic Data **5**
2.1 Overview of Basic Communication Structures . . . 5
2.2 Packet Analysis and Packet Sniffing . . . 5
2.3 Wireshark output format . . . 6
2.4 Digital Forensics . . . 7
2.5 Use-cases . . . 7

3 Visualization of Network Forensic Data **11**
3.1 Dashboard . . . 11
3.2 Methods of Visualization . . . 16

4 Analysis **18**
4.1 Problem . . . 18
4.2 Requirements . . . 19
4.3 Other Instruments . . . 19
4.4 Conclusion . . . 21

5 Design **23**
5.1 System Requirements . . . 23
5.2 Back-end . . . 24
5.3 Database Schema . . . 26
5.4 Front-end . . . 27

6 Implementation **30**
6.1 System . . . 30
6.2 Back-end . . . 30
6.3 Database . . . 40
6.4 Front-end . . . 41

7 Testing **46**
7.1 Test Definition . . . 46
7.2 Benchmarks . . . 48
7.3 Implementation Evaluation . . . 48
7.4 Enhancements . . . 49

8 Conclusion **50**

Chapter 1

Introduction

Network provider's servers handle great amounts of network traffic daily. Monitoring these amounts of traffic and all its diversity can be hard, exhausting, and nearly impossible for a human without specialized tools. On the other hand, one's failure to find and eradicate networking anomalies can lead to financial, reputation and legal damage of the organization.

This problem has led to a demand to develop an instrument that can help analyze and visualize network data. One of such tools is a dashboard, that can visualize all 4 needed use cases for faculty's network administrators: network conversations, extracted files, encrypted traffic, resolved domains.

One of the main parts of the tool is also a PCAP file parser. All the network traffic data is supplied in `.pcap` files, extracted from packet capturing software, built on the top of `libpcap`. Such programs as **Wireshark, Tshark, tcpdump** run on the server, which handles network requests, so they can capture packets and compress captured data in a `.pcap` file.

The goal of this thesis is to design and implement a system, which can provide visualizations needed to analyze network traffic in the form of 4 dashboards. Dashboards can contain a different types of visualizations and data presentations, mainly through graphs, charts and plain text. Used charts, graphs and data representations are described in Chapter 3.

The easiest and most thorough way to process `.pcap` files input is via native `libpcap` library. This library supports open-source hooks in most major programming languages and can be incorporated into any back-end server application via modern packet managers. This, however, presents an overhead of wrapping native library in one's tooling system.

One way to eradicate the aforementioned overhead is to create a specialized `.pcap` parsing library for a given set of tools, which, on the other hand, can be very time and effort consuming and can be extracted to its own thesis. Input data processing is introduced in the Chapter 2.

As of the time this thesis is written, many generic tools for visualization, data parsing and aggregation are present. The tools like **Grafana, Kibana, Graphite** are professionally used in enterprise-level applications to visualize different metrics and statistics. The tools like **influxDB, MySQL, PostgreSQL** are usually used complimentarily with the aforementioned instruments to provide a fully sustainable monitoring system. Nevertheless, the system built with these tools is not a custom build for analyzing network data and network forensic data. Many tweaks and customizations are applied in this thesis in order to provide a fuller analysis experience for the network administrator. A custom choice of charts and custom input format are also required for this tool to be usable. Further analysis of requirements, use cases and existing solutions is introduced in Chapter 4.

Chapter 5 gives a closer look at how the system is implemented, including choice of programming languages for back-end part of the application (server) and front-end part (dashboards). The architecture, UI design and database model decisions, and why they were made, are explained in that chapter.

Chapter 7 provides an overview of the application's testing. It describes two main approaches to testing—benchmarking and manual end-to-end testing. Furthermore, it evaluates the test results and provides insight into the system's weak points and how they were eradicated.

Chapter 2

Processing of Network Forensic Data

In this chapter, I introduce different formats for describing and storing network data, how they can be processed and parsed into dashboard visualization input.

2.1 Overview of Basic Communication Structures

Network communications are usually described as packets or packet flows on a network analysis level.

A packet is a group of bits that includes data plus control information. Generally refers to a network layer (OSI layer 3) protocol data unit [28].

Packet flow is a sequence of packets sent from a particular source to a particular unicast, any-cast, or multi-cast destination that the source desires to label as a flow. A flow could consist of all packets in a specific transport connection or a media stream. However, a flow is not necessarily 1:1 mapped to a transport connection [25]. This thesis identifies flows as a tuple of five fields of descriptive character: source and destination IP, source and destination port, protocol.

2.2 Packet Analysis and Packet Sniffing

Packet analysis, often referred to as packet sniffing or protocol analysis, describes the process of capturing and interpreting live data as it flows across a network in order to understand better what is happening on that network. Packet analysis is typically performed by a packet sniffer, a tool used to capture raw network data going across the wire [27].

These basic network communication structures can be stored and analyzed by various software, which mainly uses `libpcap` functionality. One of the most popular open-source tools for this cause is **Wireshark**.

Wireshark is a network analyzer. It reads packets from the network, decodes them, and presents them in easy-to-understand format [22]. However the format, in which Wireshark presents data, may be only useful for analyzing small amounts of flows and packets over short period of time. Figure 2.1 gives us a closer look at how data are represented in Wireshark.

One of the main problems with Wireshark data analysis tools is that user is not able to aggregate captured data. There is no tooling for visualization of the captured data, as well.

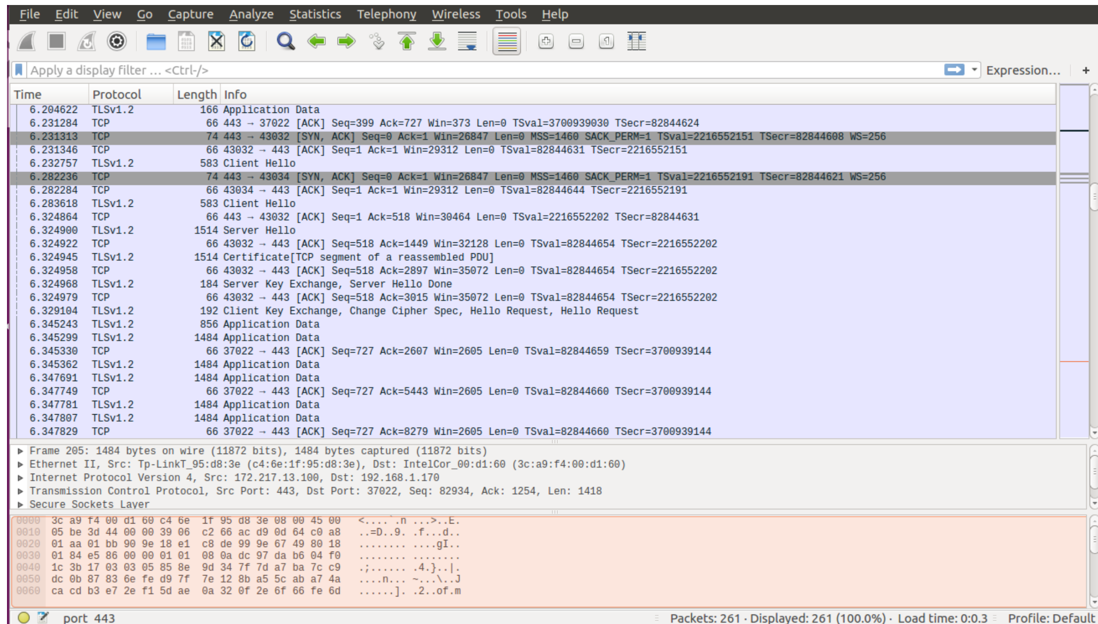

Figure 2.1: Screenshot of Wireshark GUI with packets data

The network administrator has a very deep view of what is happening inside a single flow, or inside a single packet; however, keeping in mind our use cases, this can not be scaled to the global overview of network traffic throughout a single day or even an hour. One solution to this problem is to use Wireshark only as a packet capturing software, export captured data to `libpcap` file format `.pcap` and parse it accordingly into needed input format for the dashboard.

2.3 Wireshark output format

2.3.1 libpcap Format

Wireshark uses the libpcap file format as the default format to save captured packets; this format has existed for a long time and it's pretty simple.

At the start of each libpcap capture file, some basic information is stored like a magic number to identify the libpcap file format. The most interesting information of this file start is the link-layer type (Ethernet, Token Ring, etc.) [18]. The following data is saved for each packet:

- the timestamp with millisecond resolution
- the packet length as it was „on the wire“
- the packet length as it is saved in the file
- the packet's raw bytes

2.3.2 Libraries for Working with libpcap Format

Following libraries are available to read from this format and manipulate with it [24]:

- libpcap: the origin of this file format (for UN*X based systems)
- WinPcap: Windows-based version of libpcap

More detailed overview of these libraries and their usage is provided in Chapters 5, 6.

2.4 Digital Forensics

Digital forensics is classically defined as the use of scientifically derived and proven methods toward the preservation, collection, validation, identification, analysis, interpretation, documentation and presentation of digital evidence derived from digital sources for the purpose of facilitating or furthering the reconstruction of events found to be criminal, or helping to anticipate unauthorized actions shown to be disruptive to planned operations [23].

In general, the goal of digital forensic analysis is to identify digital evidence for an investigation. An investigation typically uses both physical and digital evidence with the scientific method to draw conclusions. Examples of investigations that use digital forensics include computer intrusion, unauthorized use of corporate computers, child pornography, and any physical crime whose suspect had a computer. At the most basic level, digital forensics has three major phases [4]:

- Acquisition. save the state of a digital system so that it can be later analyzed. This is analogous to taking photographs, fingerprints, blood samples, or tire patterns from a crime scene. As in the physical world, it is unknown which data will be used as digital evidence so the goal of this phase is to save all digital values.
- Analysis: take the acquired data and examine it to identify pieces of evidence.
- Presentation: present the conclusions and corresponding evidence from the investigation.

2.5 Use-cases

For the purpose of this thesis four common use-cases of digital (network) forensics are selected. These use-cases were inspired by technical report [26].

2.5.1 Network Conversations

Network conversations provide a basic view of the network communication data. The overview of the captured packet dump should be provided to the investigator to help her with the understanding of the character of data to be analyzed. It provides a different level and type of information [26]:

- Flows: network conversation consists of a pair of flows. Each flow is identified by the flow key and has assigned statistical information. Flow key consists of transport protocol type, source and destination endpoint information.
- Hosts: based on conversation data, it is possible to compute aggregation of network communication for individual hosts or group of hosts.
- Applications: the conversation has also assigned the identified application name. The application name usually corresponds to application protocol.

Field	Description
start	The timestamp of the first packet of the conversation.
application	Identified application using a combination of port numbers and DPI methods. For known application it is the name of the service otherwise it is represented as a port number.
protocol	The name of the protocol.
client	The source IP address of the flow.
server	The destination IP address of the port.
domains	A list of domains associated with this flow.
duration	Duration of the conversation.
flowKey	The string representation of the conversation, e.g.: Udp$192.168.5.122:40080->209.85.51.222:53

Table 2.1: Format of data for Network conversations use-case

2.5.2 Extracted Files

This case considers the possibility of extraction of content from network communication in the case of unencrypted data exchange. The content is extracted on demand but it should be possible to list all available objects for extraction. It shall be possible to extract content from various plain text protocols, e.g., HTTP, FTP and SMB [26].

Field	Description
timestamp	The time of occurrence of the flow.
application	The name of the application protocol for which the content was identified and extracted, e.g., HTTP.
client	The source IP address of the flow.
contentType	The content type using MIME representation, e.g., “text/plain”.
fileType	The type of the file as observed in the file extension.
server	The destination IP address of the flow.
contentLength	The length of the content’s data.
contentName	The name of the content, e.g., the file name or full URL for HTTP objects.
exportedPath	Path to a container file with the exported object. This information can be used to localize the file itself.
fileTypeMismatch	An indication of whether content-type agrees with the file type.
flowKey	Flow information related to the event. Events are identified in packets of a flow. The flow of information helps to create a context of the issue.

Table 2.2: Format of data for Extracted Files use-case

2.5.3 Encrypted Traffic

TLS, a.k.a., SSL (Secure Sockets Layer), establishes a private end-to-end connection, optionally including strong mutual authentication, using a variety of cryptosystems [17].

Data capturing engine also collects information about encrypted communication. Currently, the TLS communication is recognized and meta information extracted from the TLS handshake, e.g, Server Name Indicator, JA3 values, associated domain names, certificate information, etc [26].

Field	Description
timestamp	Corresponds to FirstSeen field.
client	The IP address of the client.
server	The IP address of the server.
alpn	The application protocol used.
cipherSuite	The name of cipher suite selected for the channel.
duration	Duration of the conversation.
flowKey	The key string of the forward flow.
fwdRecords	The total number of TLS records in the forward (client to server) flow.
ja3Client	The fingerprint of the TLS client.
ja3Server	The fingerprint of the TLS server.
revRecords	The total number of TLS records in the reverse (server to client) flow.
serverName	Value of SNI in TLS handshake.
version	The TLS version, e.g., “Tls12”, “Tls13”

Table 2.3: Format of data for Encrypted Traffic use-case

2.5.4 Resolved Domains

The Domain Name System (DNS) is a simple query-response protocol, whose main purpose is to map domain names to IP addresses [12].

The domains table summarizes DNS queries and responses and can provide a piece of valuable information on the communication. For instance, it can be possible to apply various methods for detecting malware activity or indices of phishing attacks [26].

Field	Description
timestamp	The timestamp of the domain, i.e., the timestamp of DNS query message containing the domain.
alexa	Indication whether the domain name belongs to some of the alexa dataset.
client	The IP address of the client.
dga	A possible DGA used to generate the domain. This is computed by the DGA classifiers.
category	The category as identified using whitelist/blacklist methods.
domain	Domain name to resolve.
status	The status of the query can be NOERROR, FORMERR, SERVFAIL, NXDOMAIN, REFUSED, OTHERERROR
flowKey	The flow key of the DNS communication.
resolvedTo	The IP addresses resolved for the domain.
rtt	The round trip time in seconds.
server	The IP address of the server.

Table 2.4: Format of data for Resolved Domains use-case

Chapter 3

Visualization of Network Forensic Data

Perception is a cognitive process performed by humans in forming a mental image of domain space. In computer and information science it is, more specifically, the visual representation of a domain space using graphics, images, animated sequences, and sound augmentation to present the data, structure, and dynamic behavior of large, complex data sets that represent systems, events, processes, objects, and concepts [31].

For large volume of data and especially time-series data visualization are required so that that viewer could have a quick insight into the problem. Input data of the system designed in this thesis is both large and timed. Therefore specific visualization techniques have to be used. Because the domain of data sources is of a forensic descent, the viewer has to act quickly and precisely, thus accordingly chosen visualizations are needed. One visualization system, that can provide this quality of insight is dashboard.

3.1 Dashboard

A dashboard is a visual display of the most important information needed to achieve one or more objectives; consolidated and arranged on a single screen so the information can be monitored at a glance [7]. Example of a dashboard is displayed in 3.1.

A properly designed dashboard should follow several criteria [13]:

- Dashboard has to prioritize graphical representation of data over textual. This helps highlight relationships between data presented in a quicker and more intuitive way. It also helps with enabling of presentation of larger data sets.
- Dashboard presents only required data for the selected purpose. The designer of dashboard has to thoroughly understand the needs of the user to provide a meaningful selection of data to be presented.
- Dashboard has to fit in a single screen. Compromise between the amount of data presented and comprehensibility of the dashboard has to be found.
- Dashboard has to be intuitive so that that information can be perceived and insights gained at a glance. The dashboard should emphasize the data that matters and deserve to be spotted at the current time.

Dashboards are divided into several types [7]:

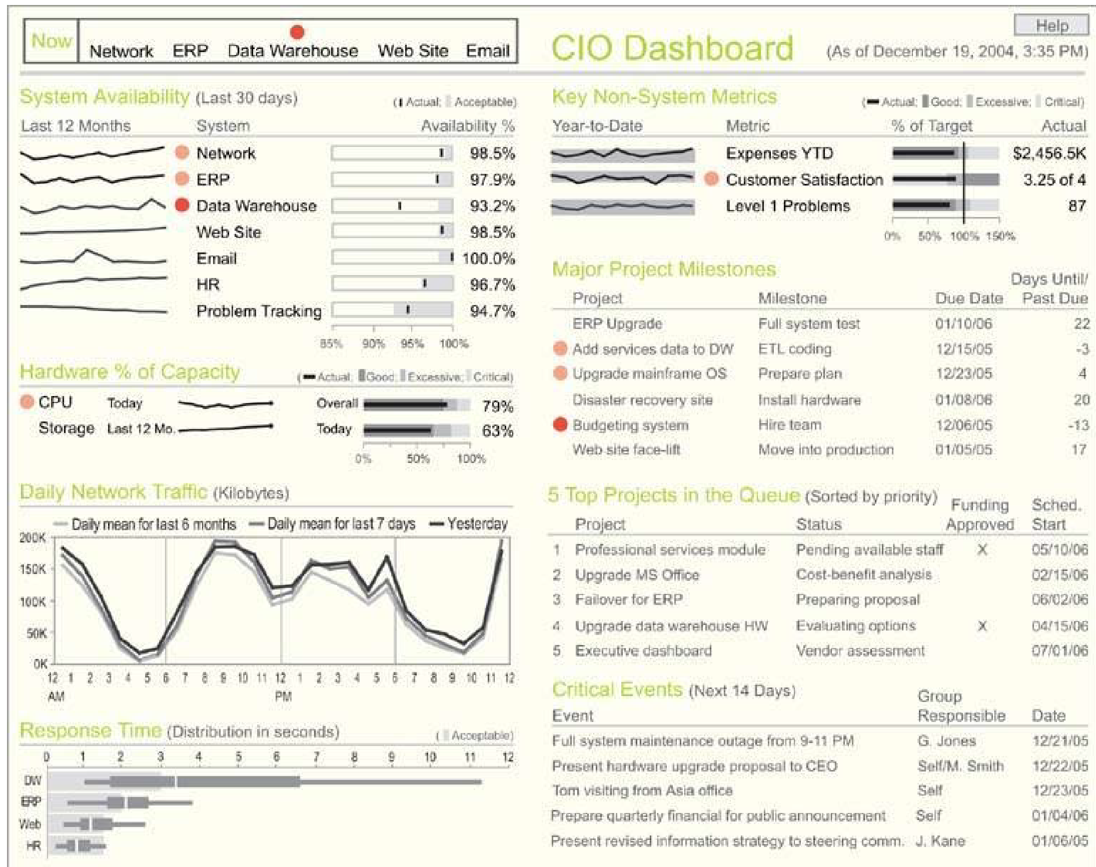

Figure 3.1: Dashboard example [7]

- **Operational** dashboards are usually designed to provide low-level data and notify users about the situations that require a response. Presentation of data is simple and data is often provided in real-time so that a quicker response can be performed.
- **Analytical (tactical)** dashboards provide aggregated data and help to analyze the data. They tend to present snapshots of data to provide a better look at the context of data.
- **Strategic** dashboards provide a summarized view over high-level data. In contrary to analytical dashboards, strategic dashboards are targeting an overview of performance and prediction of the future.

The purpose of this thesis is the development and design of analytical dashboards. Users of the dashboard developed in this thesis will have a view over snapshots of network communication data. The data will be aggregated and presented in a way useful for analyzing forensic network accidents. This purpose requires developers to select a set of visualizations accordingly, so that they will be comprehensive and graphical in terms of providing insight on the issue and data provided.

3.1.1 Visualization of Network Forensic Data

Selection of appropriate visualization types for one's requirements can be made via analysis of the underlying data. According to the data specification tables 2.1, 2.2, 2.3, 2.4 most of the provided data can be categorized into 4 groups: timestamp, numerical, categorical and textual. Therefore, the following sections will be presenting the most graphical and robust visualizations of given categories.

Line graph

A line chart (seen in Figure 3.2) is a type of chart that displays information as a series of data points called 'markers' connected by straight line segments [1]. Line charts show time-series relationships using continuous data. They allow a quick assessment of acceleration (lines curving upward), deceleration (lines curving downward), and volatility (up/down frequency).

Line chart characteristics make it useful for all the data sets provided for this thesis. Every data set has a timestamp field, numerical field and field, over which numerical data can be aggregated and categorized. Upward trend and spikes in the amount of sent bytes may interest the user, as it might be a signal of abnormal network activity.

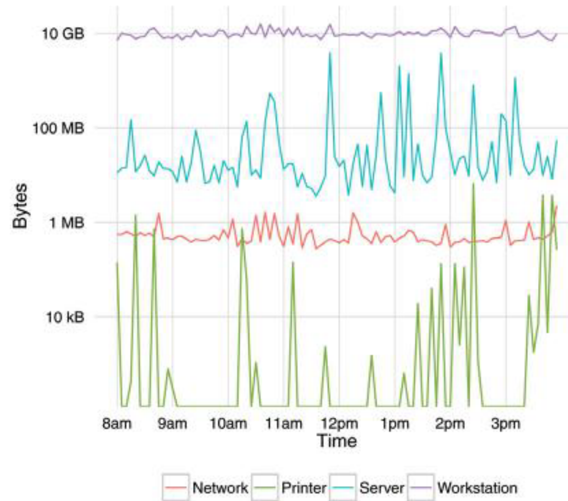

Figure 3.2: Line graph example [14]

Bar chart

A bar chart (seen in Figure 3.3) is a graph that presents categorical data with rectangular bars with heights or lengths proportional to the values that they represent. The bars can be plotted vertically or horizontally. A bar graph shows a comparison among discrete categories. One axis of the chart shows the specific categories being compared, and the other axis represents a measured value. Some bar graphs present bars clustered in groups of more than one, showing the values of more than one measured variable[3].

Bar charts are one of the most effective ways to communicate when one variable is quantitative and the other variable is categorical.[14] This particular graph can be useful for every provided data set. It can be used to visualize the amount of sent bytes grouped

by sender's IP, file type grouped by application, the content type of file grouped by client's IP, etc.

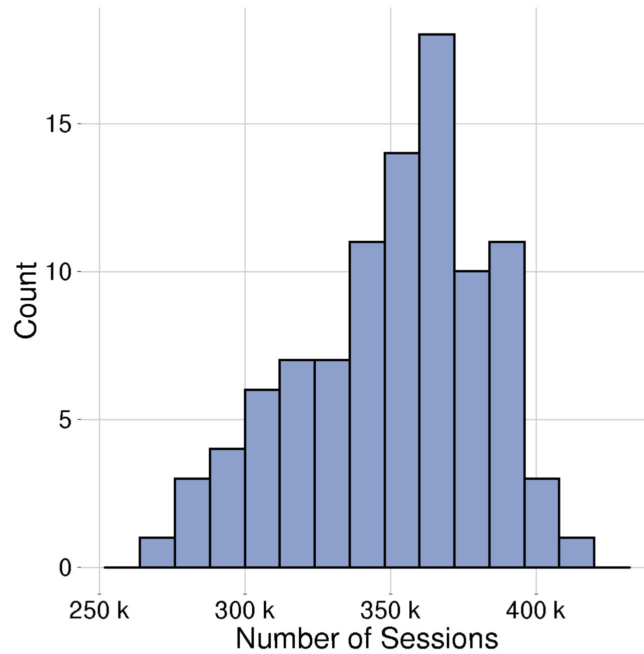

Figure 3.3: Bar graph example [14]

Sankey diagram

Sankey diagrams (seen in Figure 3.4) are traditionally used to visualize the flow of energy or materials in various networks and processes. They illustrate quantitative information about flows, their relationships, and their transformation. Sankey diagrams represent directed, weighted graphs with weight functions that satisfy flow conservation: the sum of the incoming weights for each node is equal to its outgoing weights [9].

Sankey diagrams enable showing complex processes visually, with a focus on a single aspect or resource that you want to highlight. It also makes dominant contributors or consumers stand out, which enables users to see relative magnitudes. These characteristics can be beneficial, when applied to network conversation data set, to highlight flows between different clients and servers, thus enabling the user to see the most active contributors to the captured network traffic.

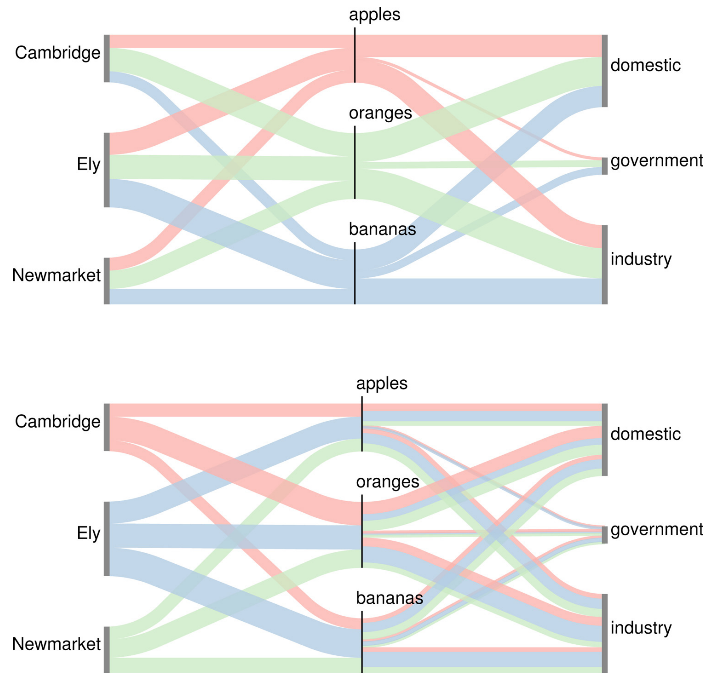

Figure 3.4: Sankey diagram example [19]

Network diagram

A drawing of a graph maps each vertex to a distinct point of the plane and each edge to a simple curve between vertices [30]. A network diagram (seen in Figure 3.5) can be either directed or undirected, weighted or unweighted, or different combinations of these options.

This type of visualization shows how things are interconnected through the use of vertices and link lines to represent their connections and help illuminate the type of relationships between a group of entities. These characteristics can be useful to help highlight communication and relations between different clients and servers in every data set.

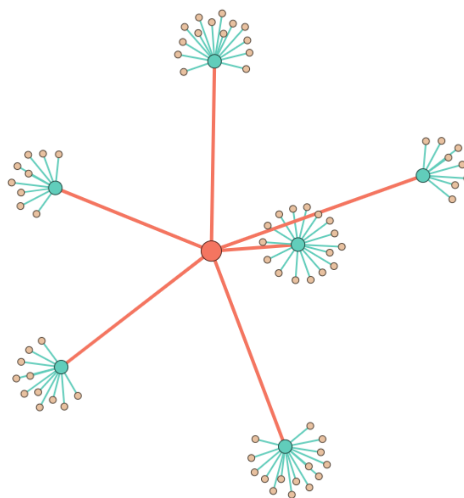

Figure 3.5: Network diagram example made by `Nivo` framework

Chord diagram

A chord diagram consists of *Arcs* and *Chords*. An arc is a segment of the circumference of the circle that is mapped to a single data entity. A chord is an area of the circle that connects two arcs together. It is possible that a chord connects an arc to itself [15]. An example of the Chord diagram is displayed in Figure 3.6.

The main functions of the Chord diagram are comparisons and relationships. Comparisons in this type of diagram are done by comparing the visible size of an arch. Another aspect of comparison is the number of chords going into and out of the arc. Relationships are visualized with chords size, color and destination.

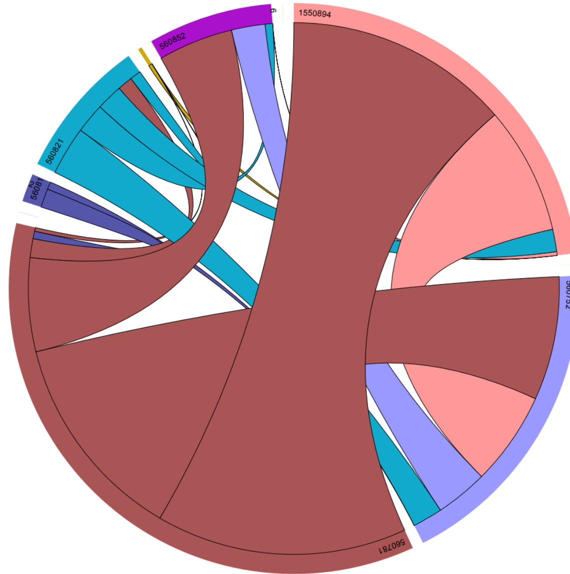

Figure 3.6: Chord diagram example [15]

3.2 Methods of Visualization

There are generally two ways of visualizing the user's data: programmable and via authoring systems. They require a different level of knowledge of how visualizations are created, how dashboards are designed, and how to program the required solution.

A more complex approach to visualization is the programmable way, which requires the user to create visualizations by themselves. Although the programmer is aided with a number of libraries and frameworks (`Nivo`, `Victoria Charts`, etc.), one is still required to program underlying systems manually. For configuration of the libraries mentioned above, the user also needs a comprehensive knowledge of visualization principles in general. This approach forces a more complex and time-consuming data visualization process but rewards the user with highly configurable and flexible charts. An even more complex yet flexible approach is to use direct Canvas rendering libraries like `D3.js` to create visualizations from scratch. It requires a more profound knowledge of programming visualizations but provides optimal solutions for the requirements.

The other approach is to use authoring systems. The learning curve is flatter, and entry-level required skills are a basic understanding of the data and query languages. This approach, however, provides the user with predefined and therefore limited visualizations,

which might not be suitable for analysis of a given domain. Moreover, some of these systems can be proprietary and therefore unaffordable by the user.

Chapter 4

Analysis

In this chapter, I will provide an analysis of the problem, requirements and contemporary tools for solving this problem.

4.1 Problem

Every entity providing Internet and/or local network access to its users is exposed to increased risk of illegal activities occurring in said network. Other problems of unmonitored network access are the high cost of hosting such a network and the possibility of damage to hardware or software used to provide the network. While several cautions, like filtering of content or DNS blacklisting, can be taken in order to prevent these problems, willing users can always find a way to evade these countermeasures and proceed with illegal activities. Moreover, the provider of the network access can be held accountable for any illegal activity undergoing. Therefore host has to possess meaningful insight into any incident, that occurred.

A network that has been prepared for forensic analysis is easy to monitor, and security vulnerabilities and configuration problems can be conveniently identified. It also allows the best possible analysis of security violations. Most importantly, analyzing a complete record of the network traffic with the appropriate reconstructive tools provides context for other breach-related events [6].

Network forensics analysis tools also can provide several benefits aside from legal concerns and security, anomalies like traffic spikes or application errors can also be studied and eradicated.

Another problem that arises from the aforementioned points is a need for specialized software or even a stack of software to assist network administrators in this cause. Some of the disadvantages of existing software are being too general to provide comprehensive insight on network forensics, being proprietary or missing analysis features. A deeper discussion of other contemporary tools can be found in Section 4.3.

4.1.1 Target Group

The following points may have already outlined the target group of the project developed in this thesis, however further details should be provided. The main target group is network administrators as providers of the network. The expected skill of a targeted user is knowledge of packet capturing software, as well as a relatively deep notion of network

technology. The most important aspects of network technology required for efficient usage of this project are packets, netowrk flows, packet layers, four use-cases (described in 2.5).

4.2 Requirements

According to the described problem, the designed system has to satisfy several requirements. The first requirement is the capability of `.PCAP` file processing. The system has to be able to produce pre-captured data in a short time so that an analyst could react to anomalies quickly. Another requirement is a suitable set of visualizations, which will provide insight into captured and processed data, however large chunks of data can be very performance-heavy and take much time to render. Therefore, data aggregation algorithms are also required to solve this problem, whatever the input may be.

While data aggregation and time bucketing are not new concepts and they are widely used among the state-of-the-art tools it is a rarely implemented feature. It is not implemented in any of the following tools, so it is logical to remove it from comparison entirely.

4.3 Other Instruments

Several contemporary network forensic tools were studied in order to find a gap and attempt to fill it. Most of them have their advantages and disadvantages that are discussed below.

4.3.1 Wireshark

Wireshark[1] is a network packet analyzer. A network packet analyzer will try to capture network packets and tries to display that packet data as detailed as possible [18]. Wireshark is actually used for this project as a bootstrap for data-source production, more detailed usage of this tool is described in Section 2.2. Although Wireshark is presented as a network forensic analysis tool, it lacks several important features. Wireshark does not provide the user with any kind of visualizations, nor a dashboard with the information needed to analyze the data. Wireshark provides a deep insight of all the packets, captured throughout its run, it also allows users to see statistics of the capture. These two features are not enough for the user to have a time-efficient analysis and do not provide a higher-level overview of the data. With the help of tools, like one developed in this thesis, the user can locate an anomaly and potential causes of it, then proceed to Wireshark to have a closer look at the data. The main Wireshark view is displayed in Figure 4.1.

4.3.2 Network Miner

NetworkMiner[2] can be used as a passive network sniffer/packet capturing tool in order to detect operating systems, sessions, hostnames, open ports, etc. without putting any traffic on the network. NetworkMiner can also parse PCAP files for off-line analysis and regenerate/reassemble transmitted files and certificates from PCAP files [11]. In many ways, NetworkMiner is similar in its comparison to Wireshark. It provides the user with statistics, packet capturing and even file extraction, however lacks quick analytical capabilities usually enabled by dashboards and visualizations. Main view of the Network Miner is displayed in Figure 4.2.

[1] https://www.wireshark.org/
[2] https://www.netresec.com/?page=networkminer

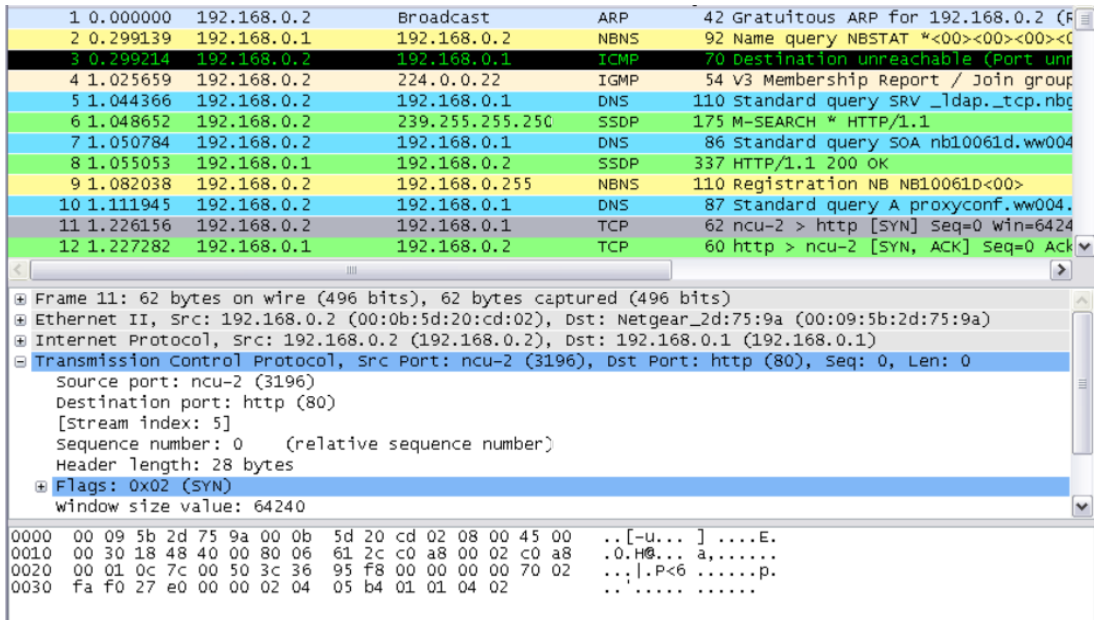

Figure 4.1: Screenshot of Wireshark's main view [18]

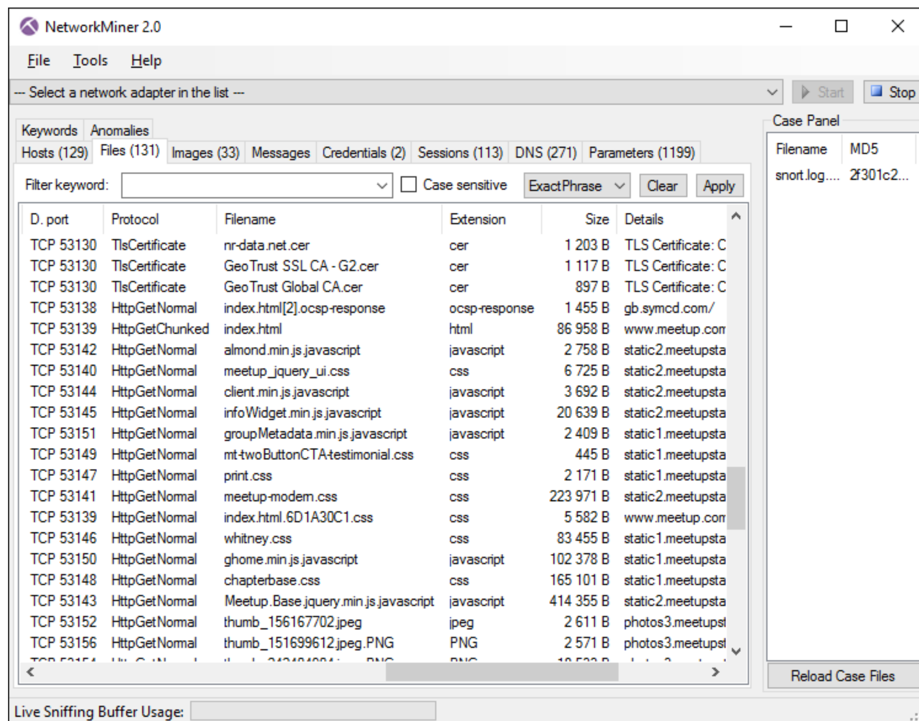

Figure 4.2: Screenshot of NetworkMiner view [11]

4.3.3 Snort

Snort[3] is an open-source system that can be used to detect flooding attacks using special rules owned by Snort. All activities are recorded on Snort are stored in a log file that records all activity on network traffic. Log files are used at this stage of the investigation to the forensic process model method to find evidence [21]. It also has one big advantage—the automated Network Intrusion Detective System (NIDS). It performs inspections on traffic and finds evidence of intrusion. It also provides basic visualizations on network data, however these visualizations are weakly connected to network forensics and are limited in analysis, that they enable. Main view of the Snort is displayed in Figure 4.3.

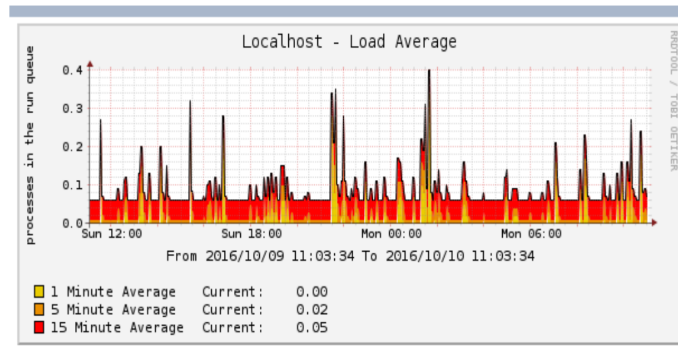

Figure 4.3: Screenshot of Snort average load visualization [21]

4.3.4 Splunk

Splunk[4] is a proprietary enterprise-level network data analysis tool. It has great capabilities of data analysis, as well as automatically triggers for different anomalies. As a cloud program, it is easily scalable and deployable. However, it is a proprietary tool, which comes at a cost, and the capabilities of Splunk are excessive in the domain of this thesis. Moreover, in dashboards still heavily rely on textual data representation and a set of visualizations is not purely suited for forensic analysis. It mostly provides health statuses of the services, response times and hit counts. Main view of the Splunk is displayed in Figure 4.4.

4.4 Conclusion

According to the problem and the comparison of the state-of-the-art tools several problems have to be solved. In order to provide a quick analytical insight into the data, a dashboard has to be designed. Furthermore, a much more comprehensive set of visualizations has to be selected. These visualizations have to represent important aspects of network forensics data. A shift from textually represented data to graphs has to be made as well.

[3] https://www.snort.org/
[4] https://www.splunk.com/

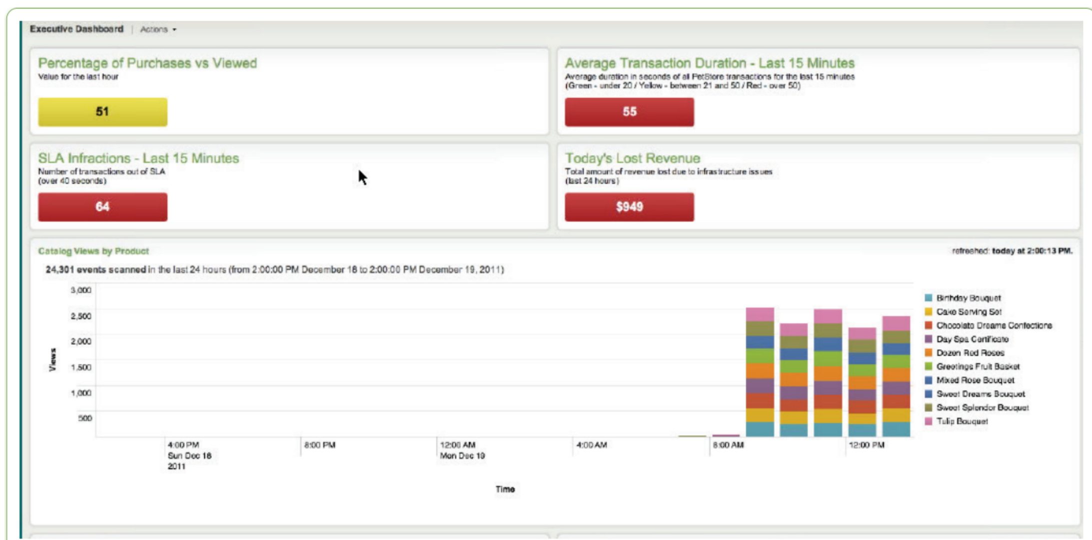

Figure 4.4: Screenshot of Splunk view [5]

Chapter 5

Design

In this chapter I will provide an overview of how the problem, defined in Analysis 4, should be solved.

5.1 System Requirements

The solution for the aforementioned problem should be a system, capable of processing network forensic data in the format of .PCAP files and visualizing them in dashboards. The proposed system should consist of two main parts: front-end and back-end.

The front-end is a website routing user actions and requests to the server with the help of API. Another responsibility of the front-end is to navigate users through all the features and possibilities of the designed system, as well as to provide them. One of the most important features is visualizing network forensic data-sets. Providing users with the possibility of selecting .PCAP data-set, which will be then sent to the server and parsed, is another vital feature, which cannot be missed. The front-end should provide the user with 4 dashboards, covering use-cases of network forensic data analysis. The dashboard has to provide a set of visualizations suitable for analysis of network forensic data, as well as enable the user to filter supplied data by time, IP, protocol, .etc.

The back-end is a server, which replies to user requests and enables all the functionality presented by the front-end. The main requirements for the back-end are:

- capability of processing .PCAP data-sets into plain table data, which can be further stored and aggregated
- providing aggregated and stored data to front-end in a way, which is the most suitable for consumption by various visualization libraries
- aggregating the data by different fields and values, so that end-users analysis is not affected or disrupted

To enable communication between two parts, an API has to be defined and implemented. The API has to provide a well defined structure of requests and responses, so that it can be reusable in different parts of the project. It also has to be precisely mapped to all the possible user interactions, so that the back-end will have a possibility of providing the functionality needed.

The database management system (DBMS) also has to be provided to the back-end. This is done for various reasons:

- back-end has to persist parsed and aggregated data, so that the user may later query or update existing data
- parsing .PCAP turned out to be a very performance costly operation, therefore results of parsing should be saved for performance improvements and as a relaxation of the parsing engine
- data aggregation can be implemented on the DBMS side to save the programmer's time

Moreover, DBMS has to be optimized to work with time, as all the provided forensic network data is time-series data.

5.2 Back-end

To achieve the aforementioned requirements back-end has to provide at least two main services: PCAP service and query execution service.

5.2.1 PCAP service

PCAP service has to enable processing of raw .PCAP data into the format, recognizable by the system. It enables data aggregation, in the form of time buckets, with user-defined parameters as well. Secondary services used to make PCAP service function properly are: the PCAP parser service, PCAP repository and time bucketing service. Architecture of the service is displayed in Figure 5.1.

PCAP parser service should take .PCAP file as input and return parsed packets table as output. Parsing functionality is being provided by external libraries, which are able to convert files into packets array directly in a program loop, or tools, which produce another human-readable file from original .PCAP file, that can be easily parsed with back-end.

Next step is storing parsed data in the database, which is done with the help of the PCAP repository. It should provide basic CRUD functionality for PCAP data.

Optional final step is to aggregate stored data with time bucketing service, based on user request. This step may be repeated with different parameters, varying in length of time bucket.

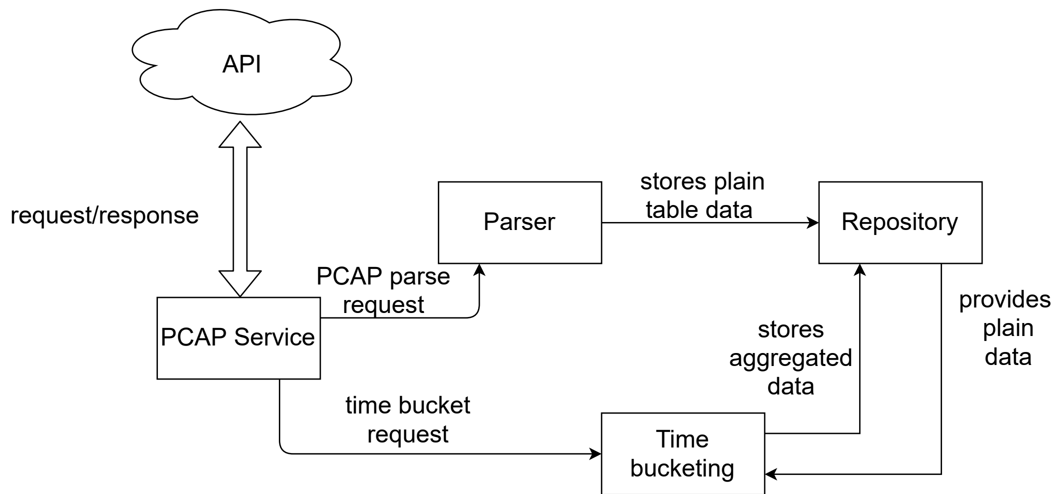

Figure 5.1: PCAP service architecture

5.2.2 Query execution service

Query execution service has to enable reading stored and aggregated from database to visualize it in the front-end. Secondary services used by Query execution to function properly are database connection template, data formatting service (transformer) and variable binder. Architecture of the service is displayed in Figure 5.2.

The input of the service is query metadata. In basic implementation, it may have a simple structure consisting only of several elements:

- data-source name, from which the data will be selected
- flag, signalising whether selected data is aggregated
- query, which will be executed on given data-source
- map of query output values to front-end visualization inputs
- formatting type
- filters

The variable binder uses input parameters to adjust query from the same input, so that predefined filters are applied. This binder is optional and depends on factors like: whether filtering is allowed in a given implementation or sole implementation of filtering (can be done in various parts of the project).

The adjusted query is then passed to database connection template. This part is usual and standard, it provides basic query routing to the DBMS and should be provided by server framework or external libraries.

The final step is passing selected data to the formatting service. It will rearrange plain data from the database in a way, suitable for consuming in the front-end. Formatting is user-defined and depends on parameters given in requests, such as formatting type and map.

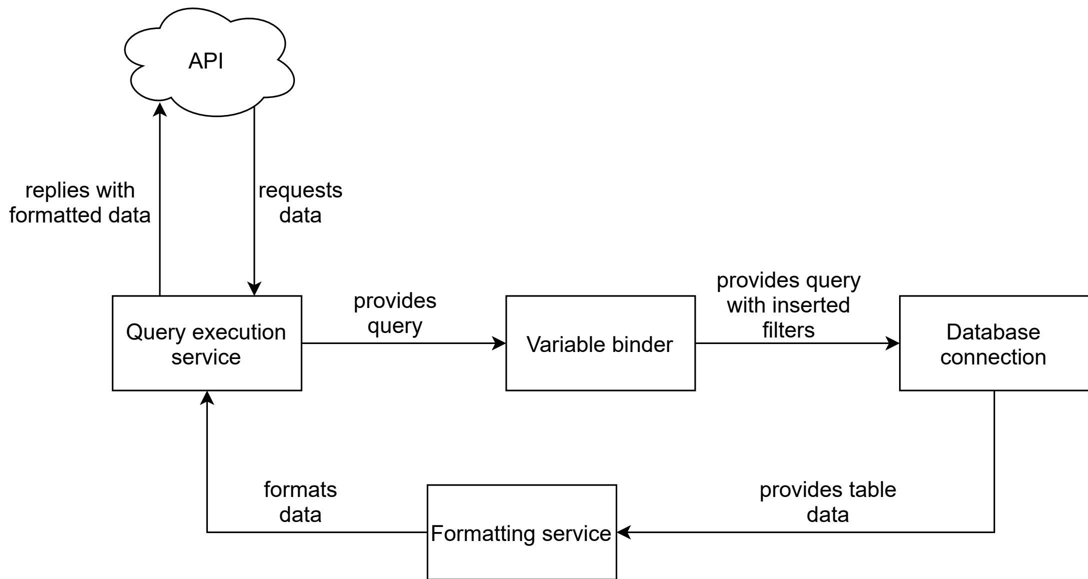

Figure 5.2: Query execution service architecture

5.3 Database Schema

The database scheme should be trivial, except for a few tweaks. These tweaks allow flexible data manipulation, such as independent creation, modification or removal of PCAP data chunks. Structure of scheme is displayed in Figure 5.3.

Several base tables have to be defined for each use case. Each base table has to represent data structures defined in Section 2.5. Metadata such as .PCAP file name has to be stored as well, so that the query execution service will be able to resolve data from a given table.

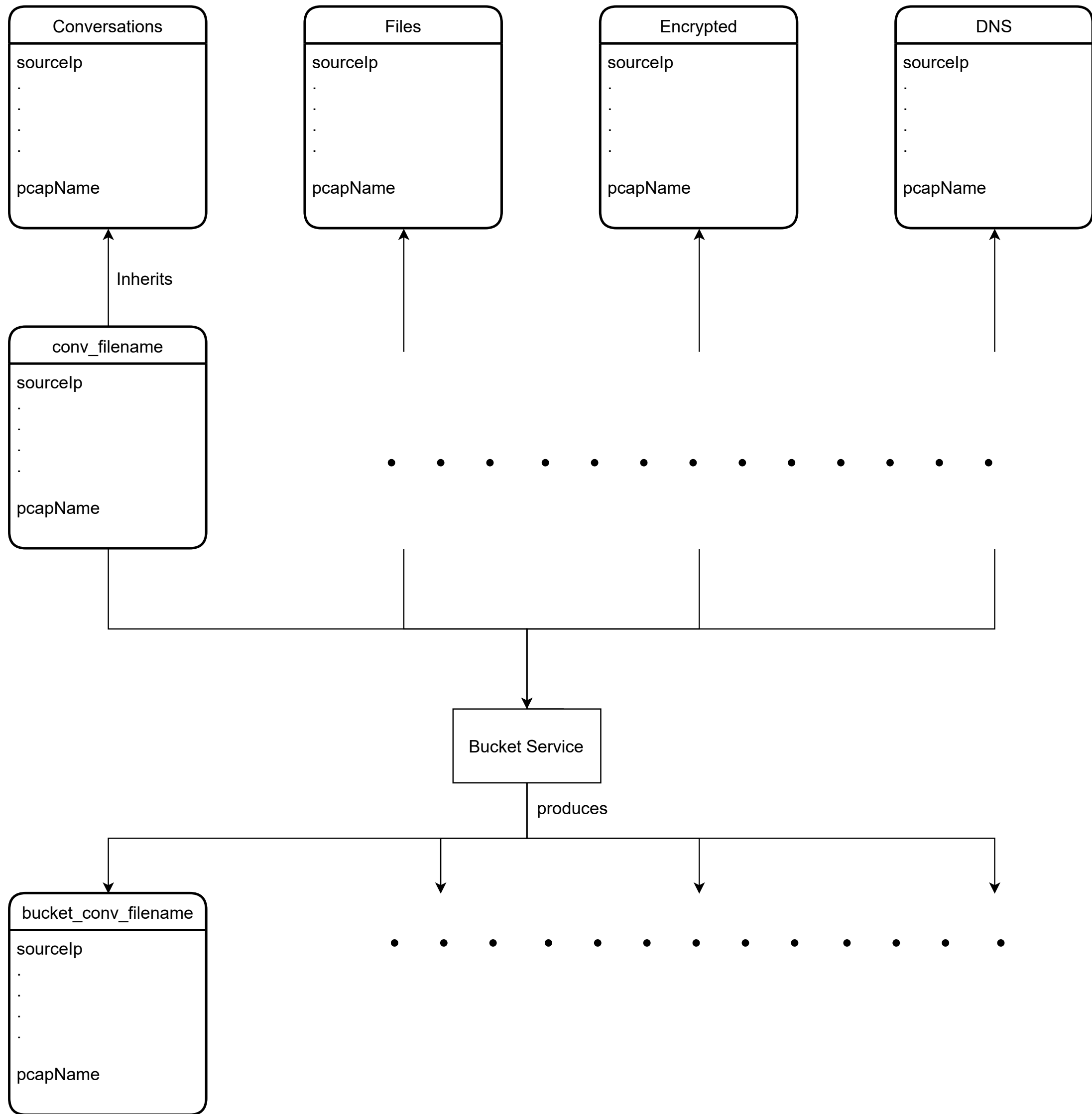

Figure 5.3: Database scheme

5.4 Front-end

Basic screens of the front-end are the PCAP submit screen and dashboard.

5.4.1 PCAP submit screen

PCAP submit screen needs to provide the user with form inputs, required for constructing meta-data for PCAP request. This meta-data are:

- PCAP type, possible values are use-cases of network forensic analysis (described in Section 2.5)
- .PCAP file

Extension of this screen would be a form with inputs for PCAP data aggregation requests. These inputs are:

- PCAP name
- bucket length

Sketch of this screen is displayed in Figure 5.4:

PCAP request

PCAP type

Conversation

PCAP file

Select File...

SUBMIT

Data aggregation request

PCAP Name

...

Bucket Length

...

SUBMIT

Figure 5.4: PCAP submit screen sketch

5.4.2 Dashboard

Basic design of a dashboard are described in Section 5.1. The sketch of the dashboard is presented in Figure 5.5. The following sections describe the numbered parts of the sketch.

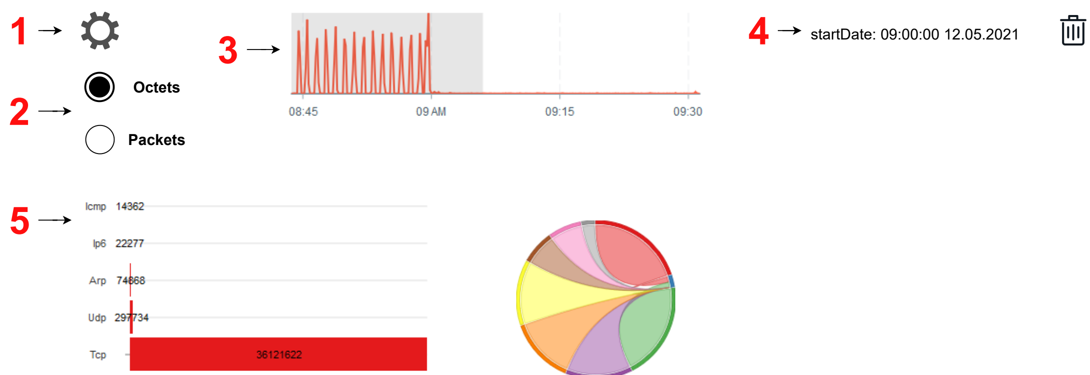

Figure 5.5: Dashboard sketch

PCAP selector (1)

PCAP selector consists of a form with inputs responsible for defining data-source for the given dashboard. Inputs are:

- PCAP name, possible values are PCAP's, which were previously uploaded and parsed by the system
- flag, signalizing whether data aggregation will be enabled for the dashboard
- bucket length, should be enabled only if the previous flag is marked as `true`

Query type selector (2)

This control is responsible for defining query type. Possible values are either packets or octets. The selected value will define the quantifier value for all the visualizations in the dashboard. For example, if the `octets` option is selected, aggregations like `sum, count` will be executed with the field `octets`.

Time selector (3)

This control provides a basic overview of the traffic density and timeline in the selected data-source. This can be implemented as a simple one-line line chart or bar chart with one bar for each time window. Time selector has to provide time filtering capabilities, which can be implemented as a brush on the chart.

Filters presenter (4)

This presenter provides an overview of all enabled filters in the form of a table and a way to modify or remove them.

Visualizations (5)

This part may greatly vary depending on the implementation and use-case selected for the dashboard. It consists of a so-called widget grid, a structure for holding all the visualizations together and in place. Every slot is taken by a single visualization, responsible for enlightening a single aspect of network forensic analysis.

Chapter 6

Implementation

In this chapter, I will provide an overview of the implementation of the system designed in the previous chapter.

6.1 System

The whole system is designed in the form of a web application, supported by all modern browsers. This web application runs in a loop of communication between three main parts: back-end server, front-end single-page application and relational SQL database. Communication is run via `REST API`, documented in `Swagger` version 2.

6.2 Back-end

6.2.1 Web Server

The back-end is implemented using several technologies, built on top of server framework `Spring`. Spring Framework is a Java platform that provides comprehensive infrastructure support for developing Java applications. The Spring Framework consists of features organized into about 20 modules. These modules are grouped into Core Container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Instrumentation, and Test, as shown in Diagram 6.1 [16].

Although, not all the modules, provided by the Spring Framework, were used in the implementation, it follows the main principles of a Spring application. Different modules and their usage will be highlighted in the following sections.

6.2.2 Programming Language

The main programming language of choice for the back-end part is Kotlin. Kotlin is a cross-platform, statically typed, general-purpose programming language. It is a Java Virtual Machine (JVM) language, which implies that the compiler will emit Java byte-code. One advantage of this fact is that, Kotlin is fully interoperable with Java language, as well as Java libraries and frameworks.

While it is being compiled to Java byte-code and executed in JVM, Kotlin is a much more modern programming language, allowing its user to enjoy syntactic sugar, `null`-safety and more. The developer of Kotlin, JetBrains, also uses byte-code to its advantage and provides extensive optimisations, which allow Kotlin to have a better performance overall.

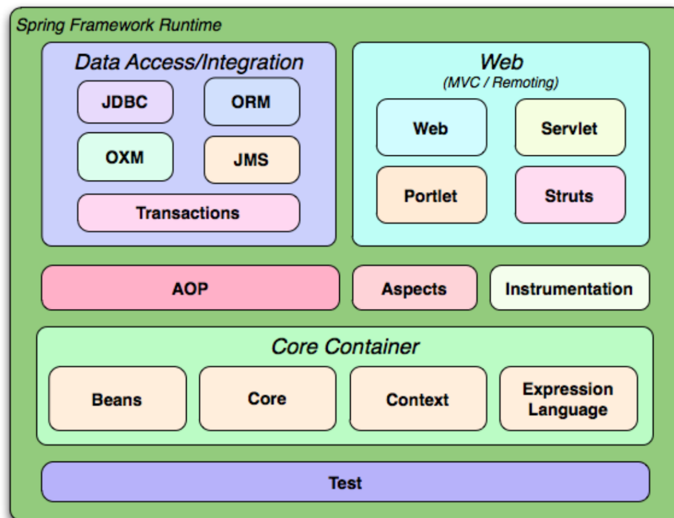

Figure 6.1: Overview of Spring framework modules [16]

Java Virtual Machine

As stated before, compiled Kotlin code runs in Java Virtual Machine, which, in its turn, makes cross-platform usage possible. JVM languages follow one exciting principle `WORA`—Write Once Run Anywhere. This means, that any operating system, which has an implementation of JVM, can run Kotlin programs without any editions.

In addition to being cross-platform, JVM offers its users one more enjoyable feature, such as Just-In-Time (JIT) compiler. The JIT compiler is a component of the run-time environment that improves the performance of Java applications by compiling byte-codes to native machine code at run time [29].

6.2.3 Assembling Web Application

This project involves a significant number of libraries, frameworks and has a complex structure by itself. To be able to assemble such a project effectively, one has to use appropriate building tools. Assemble tool of choice for this project is `Gradle`.

`Gradle` is a flexible model-driven JVM-based build tool. It acknowledges and improves the best ideas from `Make, Ant, Maven,` etc. It allows the user to define project structure, project dependencies and build tasks, with the help of simple Groovy-like Domain-Specific Language [2].

6.2.4 Web Server

The main part of the back-end is a web server, responsible for accepting HTTP requests and sending responses. The base of the web servers is a module of `Spring Framework` named `Web`. This module allows building complex web servers using different approaches and methodologies. The methodology chosen for this project is Representational state transfer (REST).

REST

REST API allows client servers communication in the form of HTTP requests and responses. The basic set of HTTP methods used for REST API is [8]:

- `GET`: Get a representation of the target resource's state
- `POST`: Let the target resource process the representation enclosed in the request
- `PUT`: Set the target resource's state to the state defined by the representation enclosed in the request
- `DELETE`: Delete the target resource's state

The basic structure of the REST request is: request URL, HTTP method, request headers and optional request body. Basic architecture of `REST` API is displayed in Figure 6.2. Request URL is mapped onto so-called endpoints, which are responsible for processing requests of the same domain. Then, the HTTP method defines the requested interaction with a given domain. These two parts are responsible for mapping request to processing entity—in case of `Spring Framework` it is a function annotated with the requested method and located in the requested controller. Requested headers are responsible for several configurations, such as type of content, sent to the endpoint, length of content, user authorization, etc. The request body is optional and is provided for `POST` and `PUT` requests, to define state change of the entity.

Another advantage of REST API is that, it is client agnostic. This implies, that any client (desktop, web or mobile application written with any framework) can successfully interact with the webserver using REST API.

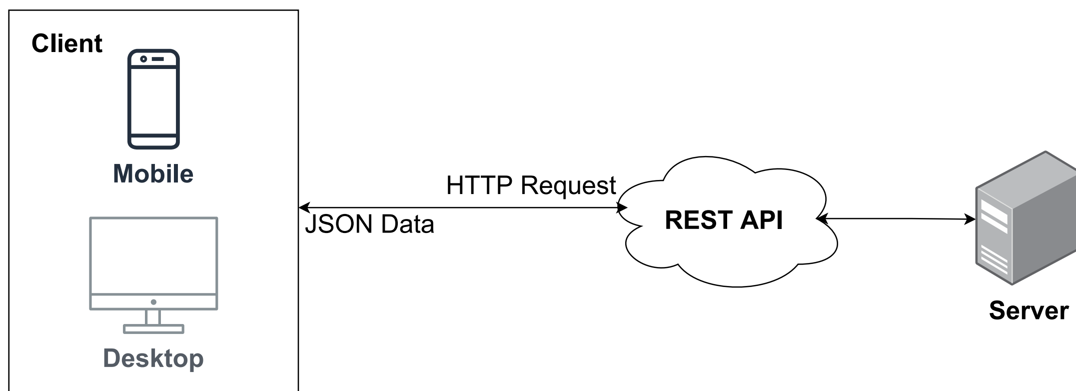

Figure 6.2: Basic REST API architecture

6.2.5 Endpoints and Controllers

This project's API has two endpoints for interactions with the server: `PCAP` endpoint and `Query` endpoint. Every endpoint has a single controller connected to it, which routes requests according to the HTTP method and path.

`PCAP` Controller

`PCAP` endpoint is responsible for providing interactions with `PCAP` files and data-sets. It is mapped to route `'/api/pcap'`. And provides four methods:

The Parse `PCAP` method is mapped to the `POST` HTTP method in the root of the controller. It is responsible for handling requests of `.PCAP` file parsing and takes two parameters: file and type of the `PCAP`. The type depends on the use-case of the given file according to the four use-cases defined in this thesis. To complete this request, the controller has to undergo several steps. The first step is to copy the uploaded file to the file system. This is done because of the limitations of `libpcap` library, which does not allow parsing of in-memory files. Then, the stored file is passed to `PCAP` service. The service processes are given file and returns object with plain `PCAP` data. Implementation of this service will be described in section 6.2.8. Then parsed data are stored in the database, with the help of `PCAP Repository`. The final step is the deletion of an uploaded file, it has been already processed and is not required for the program anymore so that the disk space can be saved.

Bucketize method is responsible for aggregating previously uploaded files by time-buckets. It is mapped to `'/bucket'` route and `POST` method. It takes only one parameter—the configuration of the following aggregation. Only one step is required to complete the request, it is passing the given configuration to the aggregation service. Implementation of aggregation service will be described in section 6.2.9.

Get all `PCAP` descriptors method is responsible for providing overview of current `PCAP` data-sets in database. It is mapped to `GET` method and to the root of the controller. It has simple database request logic, therefore will not be discussed.

Delete `PCAP` method is responsible for removing existing `PCAP` data-set from database. It is mapped to `DELETE` method and to the root of the controller. Just like the previous method, this method contains only simple database update logic and will not be described.

Query Controller

The query controller is responsible for all the interactions between the front-end and the back-end regarding query execution. It is mapped to `'/api/data'` path and has only two similar methods:

- execute query method is responsible for the execution of a single query and returning collected data. It takes only one parameter, which is the configuration of query execution. It provides very simple functionality—requesting `Query Execution Service` with given configuration. Implementation of the service is discussed in 6.2.10.
- execute query batch method is responsible for the same functionality, but with a collection of queries. It is mapped to the same `POST` method and `'/batch'` sub-route

6.2.6 Multi-threading

Previously discussed endpoints and methods can take up to 3 minutes to process requests (more about bench-marking can be learned in the chapter 7). Therefore in a multi-user environment it is impossible to execute this functionality sequentially. Because of this problem, such thing as asynchronous code execution and multi-threading has to be introduced. Multi-threading is a property of a system — whether a program, computer, or a network — where there is a separate execution point or thread of control for each process. A concurrent

system is one where computation can advance without waiting for all other computations to complete [10]. The principle of multi-threading is displayed in Figure 6.3.

In the domain of this project, several things run in parallel. The execution of endpoint methods is one of such things. Every method is run in parallel, in order to enable serving multiple users at a time. Every service and controller is specifically adjusted to be thread-agnostic, to run safely in a multi-threaded environment. Other notorious parts of the program, that have to be executed asynchronously are database and file operation. `Kotlin` provides programmers with safe parallel implementation of asynchronous file operations and `Spring Framework` does the same for the endpoints and database.

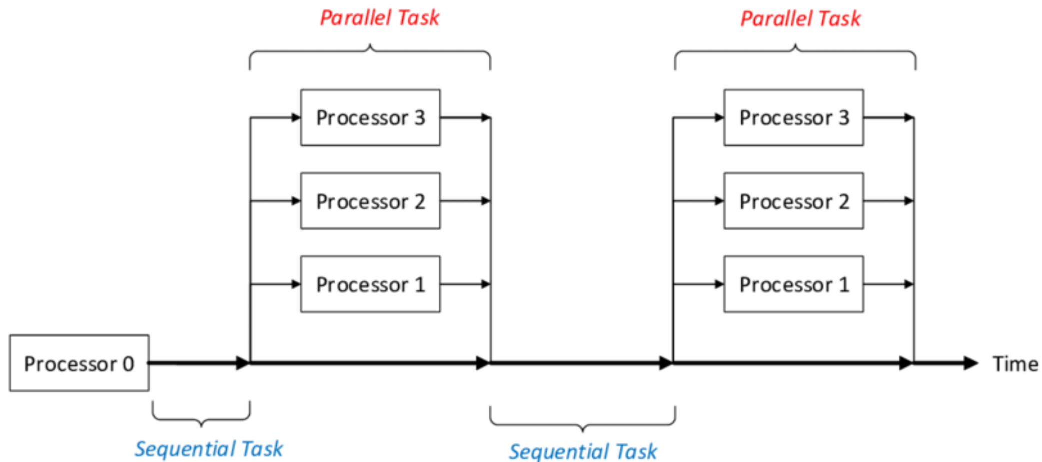

Figure 6.3: Functional principle of multi-threading. One master-thread is responsible for executing the sequential tasks and for creating the threads in parallel sections [32]

6.2.7 Dependency Injection and Inversion of Control

Another thing, that had to be introduced for this project is dependency injection. High-level modules should not depend upon low-level modules. Both should depend upon abstractions [20]. Furthermore, all the modules have to be instantiated somewhere. This is the point, where Dependency Injection (DI) comes in handy.

`Spring Framework` forces the user to use the DI. It does so by including DI into its core guidelines and by providing a core module with DI implementation. It allows programmers to write better modularized, testable code. It allows to change the implementation of a different part of the program easily and without editions, too. In `Spring Framework` this is done with help of two annotations: `@Service, @Autowired`. First defines a service, that can be injected into another part of the program. The latter defines a parameter in the constructor of a class, which has to be injected on the fly. Basic usage of DI is displayed in Figure 6.4.

```
class PcapParserResource
    private val pcapParserService: PcapParserService,
    private val bucketService: BucketService
```

Listing 6.1: Example of Dependency Injection in the project. Used services are injected into Controller automatically

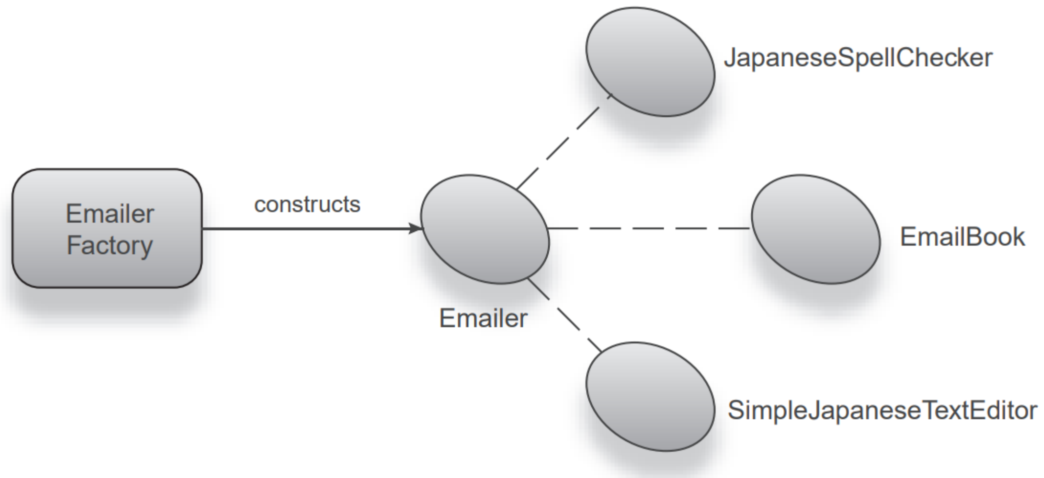

Figure 6.4: Simple example of DI with use of `Factory` Design Pattern. `EmailerFactory` constructs and assembles a Japanese `Emailer` with various dependencies [20]. Language for the `Emailer` is an Injection parameter and can be interchanged on the run.

An example of the advantages of DI is seen in the implementation of this thesis. In Listing 6.1 you can see a definition of `PCAP` endpoint. Every service is injected into the endpoint by `Spring Framework`. This allowed me to change the implementation of `bucket` service, without changing the endpoint and saved a lot of refactoring time.

6.2.8 `PCAP` Parser

`PCAP` parser is a service, which is responsible for translation of `.PCAP` file to plain data objects. It requires a single parameter as input—the name of the file. File has to be previously stored by the `File service` and has to be deleted after parsing to free up disk space. Architecture of the service is displayed in Figure 6.5.

Processing the `.PCAP` file is a simple, yet performance heavy process. It is done with the help of `libpcap` library, which is used by the tools like `Wireshark, tshark` to craft, read `.PCAP` files. This library allows users to read the file as a stream of packets. Each packet is then parsed to headers, to collect the meta-data. Different type of data is collected: source IP, destination IP, packet arrival time, packet size. It is then stored in a List data structure and persisted in the database.

`libpcap` library is written in `C` language and compiled to a dynamically linked library file, therefore it is not possible to use this type of functionality directly in `Kotlin` code. Solution to this problem is the `jnetpcap` library, which provides hooks to `Kotlin`, so that native `C` code could be used.

6.2.9 Data Aggregation

The possibility of data aggregation was discovered in the middle of writing this thesis and became one of the main features. In several use-cases data aggregation became a requirement for the system to work properly.

Aggregation of the data is done by the time field. So-called time buckets are a cluster of data between starting and ending points of the time window. These starting and ending points are defined by the user in the format of intervals.

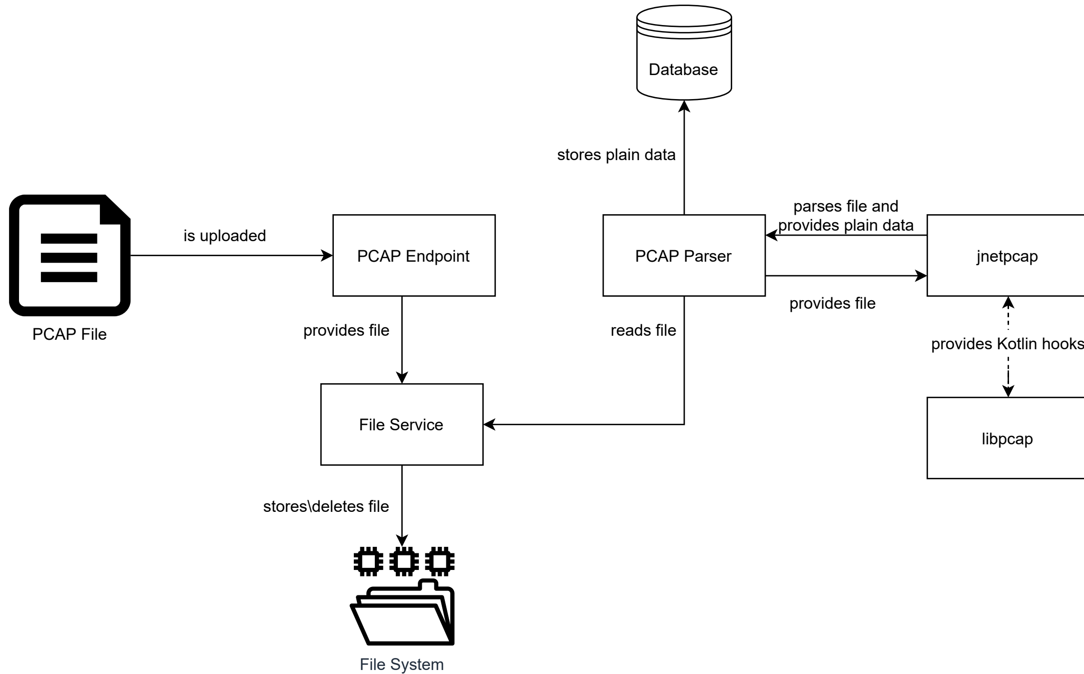

Figure 6.5: Architecture of `PCAP` parser service

The algorithm is simple yet very effective. It divides data-set into parts according to the time bucket length. Then, it scans data (packets in case of the thesis) inside the bucket and merges it together, if two data entities satisfy single constraint they are equally identified. The identification can be composed of a tuple of data entity attributes. Merging is done via a simple arithmetical sum of a quantitative field. If we apply the algorithm to the domain of this thesis, identification (primary key of data entity) is a tuple of five fields: `sourceip, destinationip, sourceport, destinationport, protocol` and the quantitative field is the number of bytes sent in a single packet. Basic explanation of the algorithm is displayed in 6.6.

The result of the algorithm is the same data-set, with aggregated unique packets inside given time intervals. This allows the user to save database space, amount of data sent from back-end to front-end up to 80% in several dense `PCAP`s. It also allows the front-end to optimize visualizations, thus allowing the largest data-sets to be visualized and large enough data-sets to be visualized with less lag and in a shorter time. `Bucket Service` is responsible for providing aforementioned functionality. This service takes two parameters as input: name of the data-set and bucket length. Then it executes query, listed in Listing 6.2, in the database:

```
select sourceip,
    time_bucket(:bucketSize, packettime) as pt,
    sum(octets), name
from :pcapName
group by sourceip,
        pt,
order by pt
```

Listing 6.2: Shorted version of query for data aggregation

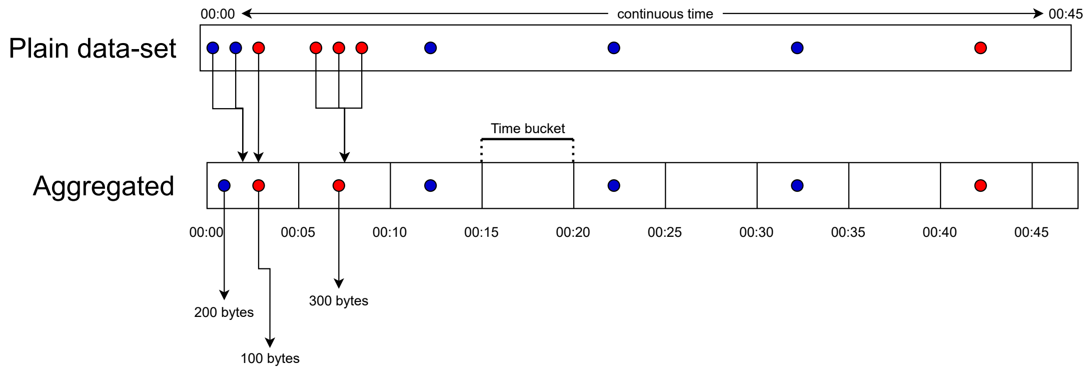

Figure 6.6: Time bucketing algorithm diagram

- Identification of data entity is made via color—same colors represent data entity with equal identification tuple: source IP, destination IP, etc.
- Each circle represents a single data entity with 100 bytes of data, unless annotated otherwise

This query executes aggregation with the help of the `time_bucket()` function and stores aggregated results in a materialized view. It selects data based on the identification tuple (`sourceip`) and aggregates the quantifier attribute (`octets`) via `sum()`. It can be later reused by the front-end to visualize data-set.

6.2.10 Query Execution Service

This service is responsible for providing, filtering and formatting data for the front-end. The input of this service is a meta-data of the query execution. It consists of several parts compiled hierarchically in `BatchDataRequestMetaData`, displayed in Listing 6.3. Classes such as `DataRequestVariables`, `DataType`, `DataMap` and `TableIdentifier` were omitted in listings for brevity. Omitted classes are key-value constant pairs, used to identify variables, format types and tables, from which the data will be selected.

```
{
  "metaDataMap": {
    "totalDestinationOctets": {
      "type": "bar",
      "mapping": {
        "qualifier": "id",
        "quantifier": "value",
        "aggregator": "ts"
      },
      "query": "select sum(octets) as value,
                     destinationip as id,
                     packettime as ts
                from :pcapName
                /*where*/
                group by id, ts
```

```
               order by ts",
      "pcapMetaData": {
        "datasourceName": "EMEA-PPP_2015-08-28-20-30-02.pcap",
        "isBucketizationEnabled": true,
        "bucketSize": 1000,
        "tableIdentifier": "conversation"
      }
    },
  },
  "variables": {
    "vars": [
      {
        "name": "startDate",
        "value": 1440793921125
      }
    ]
  }
}
```

Listing 6.3: Basic JSON structure of a data request

This data request structure is **main** communication point between front-end and back-end. Fields are defined as:

- `metaDataMap`: map of singleton parts of the query execution request
- `variables`: filters, which are applied on provided queries, described in Section 6.2.10
- `pcapMetaData`: meta-data, defining data-source upon which query will be executed
- `type`: format type, used to connect back-end output and input of front-end visualizations, described in Section 6.2.10
- `mapping`: mapping of back-end return data field names to front-end visualization input field names
- `query`: query, which will be executed in the database
- `datasourceName`: name of the base table, containing data for given query
- `isBucketizationEnabled`: flag, signalizing whether data aggregation is enabled
- `bucketSize`: size of interval for data aggregation
- `tableIdentifier`: table name, possible values are defined accordingly to four use-cases of this thesis (conversation, file, dns, encrypted)

This meta-data is then routed to several sub-services listed below. Firstly, it is routed to `Variable Service`, where the query is enriched with `WHEN` clause, in order to enable defined filters. Then it is routed to DBMS, where the query is executed. Selected data are then routed to `Transformation Service`, responsible for formatting data accordingly to a front-end input format. As it is listed in the aforementioned Listing 6.3 filtering and data aggregation parts of query execution are optional.

Filtering

`Variable Binder` is a service responsible for constructing `WHEN` clause and inserting it into the original query. Clause is constructed accordingly to the set of `variables` passed from front-end. Set can include fields like `startDate:$date`, `sourceIp:$ip`, `only_sourceIp:$ip`, etc. Left-hand value is then mapped to a column of according data-set, for example `startDate` would be mapped onto `packetTime` in `conversation` data-set. Then an operation is mapped to variable name, i.e. `startDate` would be mapped to `>=`, `endDate` to `<=`, `sourceIp` to `!=`. One exception is a variable name starting with `only`, it is used to indicate, that every other value, except for the provided with this variable has to be filtered out, therefore it maps to `=` operation. Compiled `WHEN` clause is then inserted into query, only if it has a meta-hint in it. Meta-hint looks like `/*WHEN*/` for plain query, or `/*AND*/`, if user has already predefined constraints for given query.

Given this description, query listed in 6.2.10:

```
select *
from :pcap
/*WHERE*/
```

with variables `startDate: '2012-03-19', sourceIp: 192.168.0.1` will be compiled into query listed in 6.2.10:

```
select *
from :pcap
where packetTime >= '2012-03-19'
and sourceIp != 192.168.0.1
```

Formatting

The formatting service is responsible for formatting data according to visualization type, which will consume the data. Functionality is hidden behind `Factory` design pattern to hide logic from other services. This factory currently produces four types of data formatters (also called transformers in this thesis) for four types of visualizations: bar, chord, flat (for any visualization with plain two-dimensional input), Sankey.

It takes two inputs: data to be transformed and names of return fields, so that they can be later linked in the front-end. Output is data formatted accordingly to visualization, it can be a simple List, HashMap or even a sparse matrix (Chord).

6.2.11 Database Connection

Database Connection is required for most of the back-end parts. Here data are stored, updated, aggregated and filtered. It is provided by `Spring Framework` module named `Data Access` and specifically Java Database Connectivity `JDBC`. Object–relational mapping (ORM) is another, more modern way to access data and can save a lot of programmers time to write a Database Scheme. However, this project requires flexibility in Data Access and non-traditional interactions with Database (see Listing 6.2), therefore `JDBC` is a database connection provider of choice.

6.2.12 Modularization

The whole project follows a strict module structure, and the back-end is not an exception. Implementation of complex module structure can surely bring in an overhead for a programmer, but as the project grows and changes it is a requirement for keeping the code clean and structured.

Several directories can appear multiple times in this project:

- `model`: plain classes with no logic, used to represent minimal entities of data
- `resource`: controllers for `REST API` endpoints
- `service`: classes, which provide project logic implementation, usually operate with `models` and provides functionality to `resources`

Other directories are named according to the domain of the problem, which they solve.

6.3 Database

The database Management System of choice for this project is `TimescaleDB`. It is a framework, built on top of `PostgreSQL`, allowing its users to optimize working with time-series data. One of main advantages is that it provides bucketization functionality via function `time_bucket()`.

6.3.1 Database Schema

The database schema is simple, with a base table representing and copying data models from four use-cases (see 2.5). One trick, that allows optimized usage of tables and automatically creates indices for data, is table inheritance. It also allows programmers to save time building queries, because they can be directed to the base table and will be redirected by indices to requested data. This is implemented with the help of a trigger, that creates a new child table each time a new update with a different `PCAP` name is inserted.

Table inheritance is creating a simple table name structure, as well. Each table start with a short prefix depending on use-case of underlying data: `c` for conversation, `f` for file, .etc. Every aggregated view contains the name of the table, from which aggregation was done, as well as bucket length. Example database hierarchy is displayed in Figure 6.7.

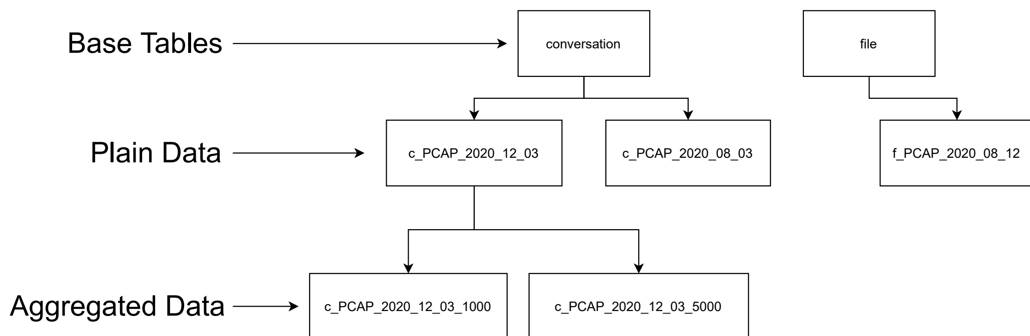

Figure 6.7: Hierarchy of database tables

6.4 Front-end

The front-end is designed as a web application running in a browser. This allows the web client to be available to reachable by the whole user group. The main technology used for creating dynamic web pages for the application is `ReactJS`.

6.4.1 React

`React` is JavaScript framework. It was originally created by `Facebook` to solve the challenges involved when developing complex user interfaces with data-sets that change over time. React moved away from the standard Model-View-Controller approach to creating simpler interfaces, made to represent data. It uses components, which are crumbs of the user interface, that can be later compiled together to create complex structures, satisfying the needs of a programmer. Another advantage of React is introduction of *Virtual Document Object Model(DOM)*. DOM is a tree structure, used by browsers to render web sites, it is parsed from HTML. Virtual DOM added a layer before real DOM, so that updates to web page structures could be done at once and only on needed components, which were updated. Therefore, a lot of rendering time is saved with this new layer of abstraction.

6.4.2 State Management

Another important part of the front-end is State Management, which is done with the help of `Redux`. The state is the backbone of UI, it defines what data will be rendered in the user browser. Applications share a global state to provide responsive interactions and back-end communication. A basic overview of the architecture of an application with a state is displayed in Figure 6.8.

Each user interaction in UI, which has any value to global application state, triggers an `Action`. It is then sent to `Reducer`, which is responsible for updating `State` accordingly to `Action` type and payload (body of the `Action`). Updated `State` then triggers a re-render on all components, which are actively using this state.

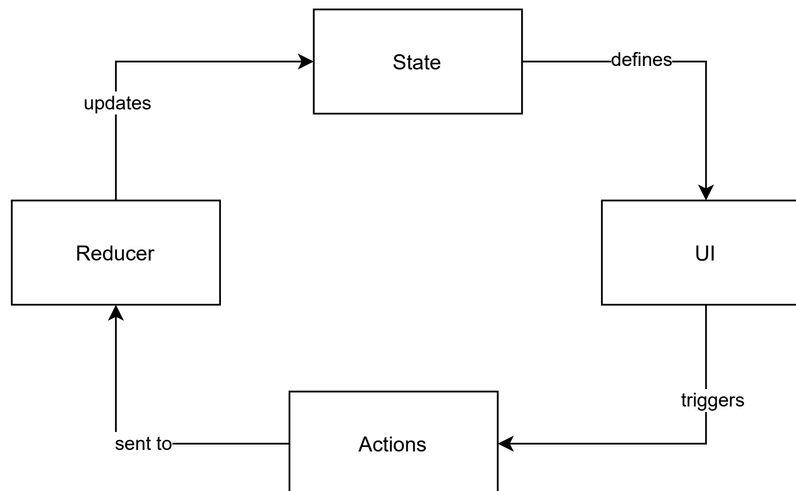

Figure 6.8: Overview of architecture of an application with state

This project has several `Stores`—slices of state, divided for separation of concerns.

- `Dashboard` state: responsible for storing user configuration of given dashboard. It includes two parts: data-source meta-data and query type, therefore logically it responds to two actions: `setDatasource` and `setQueryType`.

- `Data-source` state: responsible for storing meta-data of data-source available for user. It responds to basic `CRUD` actions: `addDatasource`, `removeDatasource`, `editDatasource`, `setDatasources`.

- `Variable` state: responsible for storing user applied filters for given dashboard. It is similar to `Data-source` store and responds to the same actions.

6.4.3 Dashboard View

The dashboard provides user with 2 main parts: Control Panel and Visualizations. Control Panel is used to apply different configurations and settings to the visualizations of the dashboard. Such configurations are data-source configurations, query type and filters. Visualizations are a set of widgets used to provide insight into different aspects of forensic network data analysis.

Figure 6.9: Screenshot of Conversations Dashboard

Control Panel

Control Panel consists of three main parts: data-source and query type selector, filter panel. It is displayed in the top part of Figure 6.9.

Data-source selector (seen in Figure 6.10) is implemented via a simple form, allowing the user to select the name of the pre-uploaded data-source. It is also possible to select, whether bucketization (data aggregation) on a given data-source has to be enabled and what size of bucket length has to be provided by the back-end. Query type selector is a simple switch between two options: `octets` and `packets`. Each of these options alters

queries, sent to the back-end, to select either sum of bytes sent in packet or the amount of packets sent.

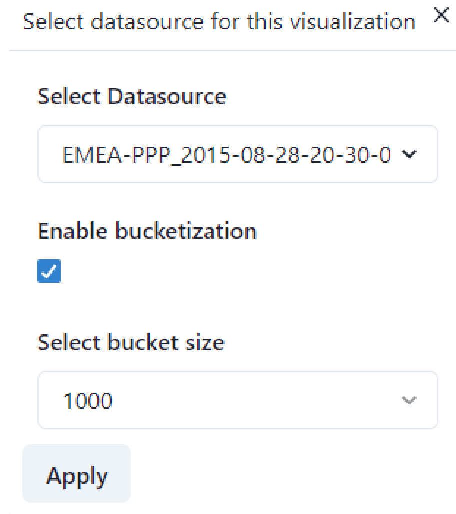

Figure 6.10: Screenshot of Data-source selector form

Filter Panel contains a line chart, representing overall information of data-set (i.e., sum of all bytes sent in a time window), and filter presenter, providing the user with an overview of currently applied filters and a way to remove the filter.

Visualizations

Visualizations are placed on a so-called widget grid, which allows responsive resizing of visualizations accordingly to different monitor sizes. Each visualization has pre-defined slots in the grid.

For example, visualizations used in `Conversations` dashboard are displayed in Figure 6.9. Descriptions follow the pattern top to bottom, left to right.

- `Text Source IP Octets`: bytes/packets sent by source IP in single time point
- `Text Destination IP Octets`: bytes/packets sent by destination IP in single time point
- `Destination to Source IP`: bytes/packets interchange between source and destination IP's.
- `Octets by Source Port`: total bytes/packets sent by source protocol.
- `Octets by Protocol`: total bytes/packets sent by protocol.
- `Octets by Source IP`: total bytes/packets sent by source IP.

Every visualization is clickable and hoverable. Click allows the user to filter out clicked elements or other elements. Hover provides a tool-tip with information about elements.

Most of the visualizations were implemented with help of a visualization framework `Nivo` [1]. It provides the user with a wide range of highly configurable visualizations. It uses the

[1] https://nivo.rocks/

same reusable component pattern as React. Every provided component has only two entry points: configuration (height, colors, legend) and data. However, every visualization requires a unique format of the provided data, therefore `data formatters` were implemented (described in 6.2.10). Nivo framework was also combined with `Victoria Charts` [2]. The latter framework is used only for a time-line line chart, which allows users to filter data by date. Victoria Charts provide a more narrow range of visualizations, but each visualization is highly configurable. Such configurations as brush were required to implement slicing of the visualization.

6.4.4 Data-sources View

Data-sources View is the second most important view in the front-end providing user controls over creation, edition and deletion of data-sources.

The view is displayed in Figure 6.11. Users can see a list of data-sources, which were previously uploaded and parsed. Every data-source can be deleted by clicking the trash bin icon next to it. Modifications to data-source can be done by clicking `Views` button. It opens bucketization form (seen in Figure 6.12), which allows users to see all data aggregations created with a given data source. Additional information like the number of packets in aggregation is also provided. This form also allows users to create new aggregation with selected unique bucket lengths.

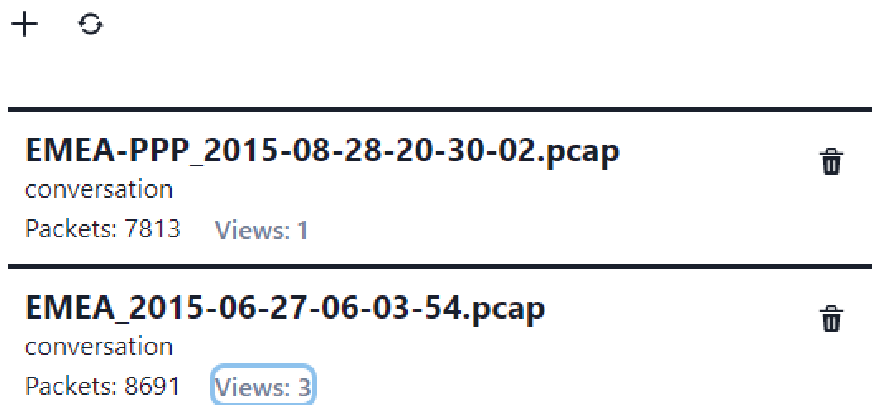

Figure 6.11: Screenshot of Data-source View

New data-source can be uploaded by clicking plus icon at the top of the view. On click a form (seen in Figure 6.13) appears. This form allows users to select the type of data-source, according to four use-cases of this thesis, and a file to be uploaded and processed.

6.4.5 Assembling

Several steps are needed to deploy a web application like this. In this project technologies like `yarn` and `nginx` are used to ease the process. `yarn` is responsible for packet management and assemble tasks, while `nginx` serves the built application at port `80`.

[2] https://formidable.com/open-source/victory/docs/victory-chart/

Bucketization options

1000: 2109

Bucket size

Bucketize

1000

Figure 6.12: Screenshot of bucketization Form and View

Add new datasource

PCAP type

Conversation

File

Choose File 2015-04-20-17-39-59.pcap

Submit

Figure 6.13: Screenshot of new Data-source creation form

Chapter 7

Testing

In this chapter, I will provide an overview of how the testing of the project was done, what results were achieved, including performance highlights, and possible enhancements.

7.1 Test Definition

Tests have one required attribute—determinism. Therefore a test plan has to be defined before the start of testing and all the tests have to follow it. Several types of testing were used in order to evaluate the quality of this project and to keep it the same way.

7.1.1 Manual End-to-End Tests

The main type of testing done was end-to-end testing. This type of testing is done manually by a programmer, by using the assembled application and observing important changes. One advantage of this type of testing is that, it asserts the correct behavior of the whole application (back-end and front-end) at once. The end-to-end testing can be very complex and time-consuming, so it is split into several domains.

PCAP Parsing

PCAP parsing test is done in several steps:

1. Open the application at `/datasources` route.
2. Click the plus icon, select `Conversation` type and upload `.PCAP` file.
3. Click `Submit`.
4. Wait for the processing to finish.
5. Check packets count, it shall be non-null and be reasonably correlated to size of file.

This test asserts that `PCAP` file upload is successful, parsing is going correctly and the database stores the parsed `PCAP`.

Data-source Manipulation

This part is responsible for the test of data-source deletion and data aggregation. The steps of this test are:

1. Open the application at `/datasources` route.
2. Click `Views` link.
3. Select unique bucket length and click submit.
4. Wait for the processing to finish.
5. Check that views count increased by one and new data aggregation has been created.
6. Return to the route.
7. Click trash bin icon.
8. Wait for the processing to finish.
9. Check that the data-source has been removed from the list.

This test asserts, that `PCAP` data-source can be deleted and new data aggregations can be created.

Dashboard

This part is responsible for the test of dashboard visualizations and data-sources linking, as well as filtering. The steps of this test are:

1. Open the application at `/dashboard/conversation` route.
2. Click gear icon.
3. Select data-source name and settings.
4. Wait for the data to be loaded.
5. Check that visualizations are loaded and they present data from selected data-source.
6. Click on any visualization and apply filter.
7. Wait for the data to load.
8. Check that visualizations are not using filtered data.
9. Return to the route.

Several settings have to be applied to cover all the test cases fully. Such settings as different data-sources and bucketization settings, different filters have to be tested as well. These tests assert that data-source is correctly linked to the dashboard and that various filters can be applied.

7.2 Benchmarks

As the system requires great performance, several benchmarks have also been run. Results of the benchmarks can be seen in Table 7.1 and Table 7.2. Two main endpoints were benchmarked: data processing and aggregation. Both of the benchmarks were tested on four of the largest data-sources.

`PCAP` processing benchmark was run via upload function in the application. Processing time also includes time to upload the file to the server, however it can be neglected as benchmarks were run on `localhost` and all the benchmarks were executed in the same conditions.Such metrics as packet count, table size and parsing time were collected. Packets count and table size are measuring the absolute size of the data-source and processing time is the main benchmark measure. The quicker data-source is processed the better performance is shown by the system.

Data-source	Raw packets	Raw size (KB)	Processing time (minutes)
A.pcap	510157	59760	2
B.pcap	360849	42328	1.3
C.pcap	321949	37806	1.2
D.pcap	304608	35708	1

Table 7.1: PCAP processing statistics

The data aggregation benchmark was run via creating a data aggregation view in the application. Such metrics as packet count, table size and compression ratio were collected. The packet count and table size are measuring the absolute size of the data-source after the aggregation and compression ratio is the main benchmark measure. The bigger the compression ratio is, the better does data aggregation function. However, this metric is highly dependent on the density of the data-source and varies between 50-86%.

Data-source	Bucket packets	Bucket size (KB)	Compression ratio
A.pcap	41215	5226	86 %
B.pcap	89766	11337	81 %
C.pcap	63320	8003	81 %
D.pcap	63101	7970	78 %

Table 7.2: Data aggregation statistics

7.3 Implementation Evaluation

One of the main purposes of the testing is to evaluate the correctness of implementation. After the test run, described in the following section, the conclusion was made, that the project satisfies most of the requirements set by the thesis assignment. Specific parts of interest for testing are:

- `PCAP` processing: possibility of processing raw `.PCAP` file to format, acceptable by the system. Acceptance criteria (design) are described in Section 5.2.1 and implementation described in Section 6.2.8. Tests have shown, that the project is fully capable of processing any `.PCAP` file.

- Data-source manipulation: the capability of creating new data-sources from PCAP data and further manipulation with them such as deletion. Tests have shown, that the project is capable of creating new data-source and deleting existing ones.
- Data aggregation: the optional possibility of aggregating existing data. The system is capable of creating data aggregation of unique bucket length in the range of `integer` data type, therefore fully accepted.
- Query execution: supplies data to the dashboard visualizations. Acceptance criteria are described in Section 5.2.1and implementation described in Section 6.2.8. It is fully functional according to **Design**.
- Visualization correctness: data supplied from data-source should be precisely displayed in graphs. According to tests—fully functional.
- Visualization responsiveness: graphs are showing additional information on hover, allow the user to filter data on click, are dynamically resized with browsers window.
- Filtering: data are filtered and displayed in visualization according to applied filters.
- Use-cases coverage: only one use-case is covered in implementation, however the base of the system is prepared to interact with any other use-case with minimal effort. The project is implemented with a generic approach so that any new requirements could be implemented with a minimal number of editions and minimal time.

The project has full implementation of the base system, which is open to any modifications, due to its generic implementation. One path to *Conversation* use-case is fully implemented as well, and can serve as an example for future implementation of other use-cases.

7.4 Enhancements

According to the completed tests, two flaws in the system were found. Three dashboards were not finished and native `libpcap` library linking is sometimes lagging.

Dashboard's readiness can easily be improved by a small number of adjustments, due to the system's generic architecture. The only thing left is to design the dashboard view and implement it in the front-end.

The `libpcap` library linking can be improved by contributing to the `jnetpcap` project or by changing the implementation of the PCAP parser to use a different library. Issue has already been created and is tracked at the GitHub page of the `jnetpcap` library in **Issues**[1] tab.

[1] https://github.com/ruedigergad/clj-net-pcap/issues/13

Chapter 8

Conclusion

The goal of this thesis was to implement a system for network forensic data analysis. The implemented system enables users to process captured `.PCAP` files to data-sources, aggregate and compress them and visualize results. The system is implemented in the form of a dashboard.

Based on the comparison of the state-of-the-art network forensic tools a set of insightful visualizations was chosen and designed for each use-case. Furthermore, the features like filtering by IP addresses, protocols, port numbers and time were added. The data aggregation feature was enhanced to support multiple aggregations on a single data-source. This feature is required to visualize larger data-sources. A generic base system in a form of the back-end was developed to support polymorphic clients and to provide easily extensible functionality for further improvements. Such technologies as time-series databases were studied, as well. Optimized for read and write operations database scheme was designed and implemented, with help of the table inheritance.

Dashboard design principles were studied and used to design a compact comprehensive dashboard. Several visualization libraries and frameworks were compared and the most suitable one was chosen to render graphs.

This work can be developed further by implementing a heuristic system of automatic creation of data aggregations. Another useful feature would be adding more file input formats, such as `.CSV`. Additionally, other use-cases such as *Extracted Files*, *DNS* and *Encrypted Traffic* can be covered more deeply and comprehensively.

The final application was tested manually with end-to-end tests and its performance was evaluated with several benchmarks. The project is public on GitHub[1].

[1] https://github.com/Fonifan/NetDash

Bibliography

[1] Andreas, B. *Experimental psychology.* Wiley, 1965.

[2] Berglund, T. and McCullough, M. *Building and testing with Gradle.* „ O'Reilly Media, Inc.“, 2011.

[3] Bhalla, C. *Audio-visual Aids in Education.* Atma Ram, 1963.

[4] Carrier, B. Open source digital forensics tools. *The legal argument.* Citeseer. 2003.

[5] Casey, E. *Handbook of digital forensics and investigation.* Academic Press, 2009.

[6] Corey, V., Peterman, C., Shearin, S., Greenberg, M. S. and Van Bokkelen, J. Network forensics analysis. *IEEE Internet Computing.* IEEE. 2002, vol. 6, no. 6, p. 60–66.

[7] Few, S. *Information dashboard design: The effective visual communication of data.* O'Reilly Media, Inc., 2006.

[8] Fielding, R. and Reschke, J. *Hypertext transfer protocol (http/1.1): Semantics and content.* rfc 7231, June, 2014.

[9] Froehlich, P. Interactive Sankey Diagrams. In: *IEEE Symp. on Information Visualization.* 2005, p. 233.

[10] Galvin, P. B., Gagne, G., Silberschatz, A. et al. *Operating system concepts.* John Wiley & Sons, 2003.

[11] Hjelmvik, E. *NetworkMiner - The NSM and Network Forensics Analysis Tool.* Available at: https://www.netresec.com/?page=networkminer.

[12] Hoffman, P., Sullivan, A. and Fujiwara, K. DNS terminology. *RFC 7719.* 2015.

[13] Hynek, J. *Impact of subjective visual perception on automatic evaluation of dashboard design.* 2019. Dissertation. Brno University of Technology.

[14] Jacobs, J. and Rudis, B. *Data-driven security: analysis, visualization and dashboards.* John Wiley & Sons, 2014.

[15] Jalali, A. Supporting social network analysis using chord diagram in process mining. In: Springer. *International Conference on Business Informatics Research.* 2016, p. 16–32.

[16] Johnson, R., Hoeller, J., Donald, K., Sampaleanu, C., Harrop, R. et al. The spring framework–reference documentation. *Interface.* 2004, vol. 21, p. 27.

[17] Khare, R. and Lawrence, S. *RFC2817: Upgrading to TLS Within HTTP/1.1*. RFC Editor, 2000.

[18] Lamping, U. and Warnicke, E. Wireshark user's guide. *Interface*. 2004, vol. 4, no. 6, p. 1.

[19] Lupton, R. and Allwood, J. Hybrid Sankey diagrams: Visual analysis of multidimensional data for understanding resource use. *Resources, Conservation and Recycling*. 2017, vol. 124, p. 141–151. DOI: https://doi.org/10.1016/j.resconrec.2017.05.002. ISSN 0921-3449. Available at: https://www.sciencedirect.com/science/article/pii/S0921344917301167.

[20] Martin, R. C. The dependency inversion principle. *C++ Report*. 1996, vol. 8, no. 6, p. 61–66.

[21] Mualfah, D. and Riadi, I. Network Forensics For Detecting Flooding Attack On Web Server. *International Journal of Computer Science and Information Security*. LJS Publishing. 2017, vol. 15, no. 2, p. 326.

[22] Orebaugh, A., Ramirez, G. and Beale, J. *Wireshark & Ethereal network protocol analyzer toolkit*. Elsevier, 2006.

[23] Palmer, G. Report from the first digital forensic research workshop (DFRWS). *Tech. Rep.* Air Force Research Laboratory, Rome Research Site. 2001.

[24] Perry, D. *Libpcap File Format*. Last changed: 2015-08-23. Available at: https://wiki.wireshark.org/Development/LibpcapFileFormat.

[25] Rajahalme, J., Conta, A., Carpenter, B. and Deering, S. *RFC3697: IPv6 Flow Label Specification*. RFC Editor, 2004.

[26] Ryšavý, O. and Hynek, J. Visual Representation of Network Forensic Data. *Technical Report no. FIT-TR-2020-07*. Faculty of Information Technology, Brno University of Technology. 2020.

[27] Sanders, C. *Practical packet analysis: Using Wireshark to solve real-world network problems*. No Starch Press, 2017.

[28] Stallings, W. and Van Slyke, R. *Business data communications*. Upper Saddle River, N.J.: Prentice Hall, 2000.

[29] Suganuma, T., Ogasawara, T., Takeuchi, M., Yasue, T., Kawahito, M. et al. Overview of the IBM Java just-in-time compiler. *IBM systems Journal*. IBM. 2000, vol. 39, no. 1, p. 175–193.

[30] Tamassia, R. *Handbook of graph drawing and visualization*. CRC press, 2013.

[31] Telea, A. C. *Data visualization: principles and practice*. CRC Press, 2014.

[32] Wolf, J., Baumgärtner, D., Rossi, R., Dadvand, P. and Wüchner, R. *Contribution to the Fluid-Structure Interaction Analysis of Ultra-Lightweight Structures using an Embedded Approach*. January 2015. ISBN 978-84-943307-6-6.