



**University of
Zurich**^{UZH}

Kronos - A Forensic Network Data Inspection Tool

A project in collaboration with the Zürich Cantonal Police

*Lenz Baumann & Cyrill Halter
Zurich, Switzerland
Student ID: 10-741-577, 13-928-171*

Supervisor: Bruno Rodrigues, Eder Scheid, Steffen Görlich (KAPO)
Date of Submission: 13.01.2020

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland



Abstract

The field of digital forensics has become increasingly important with the rapid development of technology and the integration of computer systems into every aspect of our daily lives for both law enforcement and industry. The subfield of network forensics, in particular, has become essential with the convergence of communication and applications to the internet. This report describes a forensic network data inspection tool developed in collaboration with the Zürich Cantonal Police (KAPO ZH) within the scope of a Master Project at the University of Zürich. Use cases and requirements are assessed from conversations with the employees at the Digital Forensics group at KAPO ZH. A system is then outlined that meets these requirements while at the same time maintaining efficiency and scalability by integrating tightly with the existing infrastructure at KAPO ZH. An implementation is described that uses modern programming languages and frameworks and an agile development process to fit the specified topology. The system is then evaluated on a functional and qualitative level. The evaluation shows that the application fulfills the use cases put forward by KAPO ZH. It enables insight into the network data that can reveal usage patterns, applications used and devices used. Furthermore, the evaluation shows that the developed system is highly usable and maintainable.

Zusammenfassung

Der Bereich der digitalen Forensik hat mit den rasanten technischen Entwicklungen der letzten Jahrzehnte und der damit einhergehenden Integration von Computersystemen in jeden Aspekt unseres täglichen Lebens sowohl für die Strafverfolgung als auch für die Industrie zunehmend an Bedeutung gewonnen. Insbesondere der Teilbereich der Netzwerk-Forensik ist mit der Verschiebung von Kommunikation und Applikationen ins Internet immer wichtiger geworden. Der vorliegende Bericht beschreibt ein Tool zur forensischen Untersuchung von Netzwerkdaten, welches in Zusammenarbeit mit der Kantonspolizei Zürich (KAPO ZH) im Rahmen eines Masterprojektes der Universität Zürich entwickelt wurde. Alle Anforderungen und Einsatzmöglichkeiten wurden anhand von Gesprächen mit den Mitarbeitenden der Abteilung Digitale Forensik der KAPO ZH ermittelt. Das daraus hervorgehende System soll all diesen Anforderungen gerecht werden und gleichzeitig durch seine enge Integration in die bestehende Infrastruktur der KAPO ZH effizient einsetzbar und einfach skalierbar sein. Die Implementierung erfolgt durch den Einsatz moderner Programmiersprachen und Frameworks sowie durch einen agilen Entwicklungsprozess und wendet die spezifizierte Topologie an. Die Applikation ist in der Lage Netzwerkdaten, Nutzungsmuster, genutzte Anwendungen und verwendete Geräte aufzuzeigen. Eine Evaluation anhand des Anforderungsprofils zeigt, dass die Applikation die von der KAPO ZH vorgeschlagenen Anwendungsfälle erfüllen kann. Darüber hinaus zeigt sich, dass das entwickelte System in hohem Maße nutz- und wartbar ist.

Acknowledgments

We would like to express our greatest appreciation to Steffen Görlich and his team from KAPO ZH, who offered continued support and constant encouragement during our work and made us feel comfortable right from the start. We would also like to thank Bruno Rodrigues, whose ideas, suggestions and comments helped along our work and steered us in the right direction. Finally, we would like to thank Professor Burkhardt Stiller for his support and the opportunity to work in a unique field and with such interesting data.

Contents

Abstract	i
Zusammenfassung	iii
Acknowledgments	v
1 Introduction	1
1.1 Motivation and Description of Work	1
1.2 Report Outline	2
2 Background and Related Work	3
2.1 Digital Forensics	3
2.2 Network Forensics	3
2.3 Network Traffic and Network Traffic Monitoring	4
2.4 Network Traffic Encryption	4
2.5 Pseudorandom Number Sequences	4
2.5.1 Entropy	5
2.5.2 $\tilde{\chi}^2$ -Square Test	5
2.5.3 Arithmetic Mean	6
2.5.4 Monte Carlo Value for Pi	6
2.5.5 Serial Correlation Coefficient	6
2.6 OS-level Virtualization System	6
2.7 Web Application	7

2.8	Related Work	7
2.8.1	ENT	7
2.8.2	Tranalyzer	7
2.8.3	CyberChef	7
2.8.4	Wireshark	8
2.8.5	Elasticsearch	8
2.8.6	Moloch	8
2.8.7	Docker	8
3	Architecture	11
3.1	Naming	11
3.2	Design Considerations	11
3.3	Requirements	12
3.3.1	Project Goals	12
3.3.2	Digital Forensics at KAPO ZH	13
3.3.3	Functional Requirements	14
3.3.4	Quality Requirements	15
3.4	Assumptions	15
3.5	System Topology	15
4	Implementation	19
4.1	Kronos Client	19
4.1.1	Languages and Frameworks	19
4.1.2	Design Patterns	20
4.1.3	Implementation Details	21
4.2	Kronos Server	22
4.2.1	Languages and Frameworks	22
4.2.2	Design Patterns	23
4.2.3	Implementation Details	24

4.3	Processes	25
4.3.1	Login	25
4.3.2	Data Retrieval	25
4.3.3	Link Generation	27
4.4	Productive Environment and Deployment	28
4.5	Development Process	29
4.6	Research Into Encryption Detection	29
5	Evaluation	33
5.1	Evaluation Data Set	33
5.2	Use Cases	34
5.2.1	Insights into applications or devices in use	34
5.2.2	Automated identification of unencrypted network traffic	38
5.2.3	Identify patterns or anomalies	39
5.3	Evaluation of Requirements	41
5.3.1	Functional Requirements	41
5.3.2	Quality Requirements	43
6	Summary and Future Work	47
6.1	Future Work	47
	Bibliography	52
	Abbreviations	53
	List of Figures	53
	List of Tables	56
A	Kronos Github Repository	59
B	Contents of the CD	61
C	SUS Evaluation Form	63

Chapter 1

Introduction

The field of digital forensics has become increasingly important with the rapid development of technology and the integration of computer systems into every aspect of our daily lives. It has become an essential task for law enforcement authorities in the investigation not only of crimes committed in the digital domain, but also any other crime that indirectly involves a computer. Since most of our work and communication takes place on a digital device, the data stored on it can provide extensive information about our actions, social environment and habits.

Network forensics, a subfield of digital forensics, deals with the recovery of data from networked communication, taking place mostly over the internet. The growth of the internet and the relocation of many applications into the web have led to this area encompassing a significant part of the digital forensics field overall. Recent developments, however, have complicated the task. The emergence and spread of encryption on different levels of the network have made the contents of networked communication inaccessible to large parts. This issue may become even more serious with the encryption of network management protocols, *e.g.* DNS over HTTPS (DoH).

1.1 Motivation and Description of Work

The department of digital forensics at the Zürich Cantonal Police (KAPO ZH) processes network traffic data from network surveillance of persons of interest. Network data can be collected (retrieved on-site) or obtained from an Internet Service Provider (ISP), and is then processed by a forensic specialist. He will search the data for services used by the person of interest, such as messaging tools, financial services, and e-mail services. The forensic specialist will also look for any further interesting information in an explorative manner. An outcome of such an analysis is, for example, a summary of findings that can provide additional insights to support an investigator, serving as an indicative point to start or close investigations. In case the information is later used in court, its provenance and sources need to be evident within the analyzed network data.

The main issue with the current situation at KAPO ZH is that a lot of manual labor is involved in gaining insights into the network traffic. While certain tools are in use, they are not tailored to the very specific use cases at hand. This leads to a very high expenditure of time and, in some cases, results in tasks having to be taken care of the outside of the existing tooling, *e.g.* using bash scripts. Most importantly, time is a very limited resource for KAPO ZH and any increase in efficiency or effectiveness is desirable.

The system implemented within the scope of this master project aims to simplify and improve KAPO ZH's network forensics process by attempting to automate common cumbersome analytic tasks. In addition to this, it aims to enable analytics that were not possible within the existing framework in use today.

1.2 Report Outline

The rest of this report will be structured as follows: Chapter 2 will elaborate on the scientific background of the concepts used in our system and introduce work related to our topic. Chapter 3 will discuss the requirements for our application and define the system's topology at a high level. The actual implementation will be explained in detail in Chapter 4. Chapter 5 will evaluate whether the goals set were met and to what degree. Finally, Chapter 6 will summarize the work and conclude this report.

Chapter 2

Background and Related Work

In this chapter, different technologies as well as technical terms and concepts are elaborated on and explained within the context of this work.

2.1 Digital Forensics

Digital forensics is a broad field of tasks including the monitoring, investigation and inspection of network traffic (*cf.* Section 2.2) and the extraction and recovery of data from various digital assets such as mobile devices and computers or databases [1][2][3]. Digital forensics is composed of subfields, each specialized in a specific area such as Network forensics, Mobile Forensics, Computer Forensics and Forensic Data Analysis [1]. Within the context of this work, network forensics is the most relevant.

2.2 Network Forensics

Network forensics focuses on monitoring and the analysis of network traffic. The prime goals involve the monitoring of a network for anomalous traffic and the identification of intrusions and the reassembly of transferred files, the search for keywords and the parsing of human communication such as emails or chat sessions [4].

To inspect network traffic, a dissector is used. “A network traffic dissector is a program, that allows decomposing a data package following a certain set of characteristics.”[5] With the help of a dissector, a certain protocol can be understood and its packets decomposed [6]. Network traffic dissector capabilities are offered by tools such as Wireshark [7] (*cf.* Section 2.8.4) and Moloch (*cf.* Section 2.8.6).

2.3 Network Traffic and Network Traffic Monitoring

Network traffic is mostly encapsulated in network packets with a well-defined structure, that, in the context of network forensics, (*cf.* Section 2.2) can be monitored, inspected and analyzed at different levels of granularity (*e.g.* byte, packet, flow or connection). Such network traffic inspection can be either done live (live monitoring and inspection) or ex-post (inspected later) and can be differentiated between passive (only capturing and inspecting) and active (including injection and altering of packets) network monitoring.

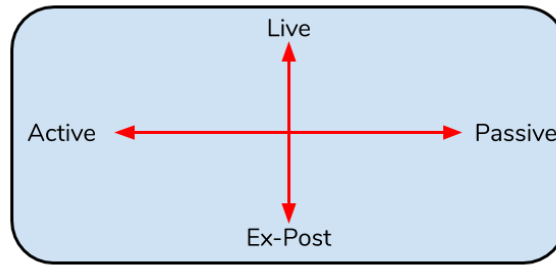


Figure 2.1: The two dimensions of network traffic monitoring: Active vs. Passive monitoring and Live vs. Ex Post monitoring [8].

2.4 Network Traffic Encryption

Network traffic can be encrypted using network traffic protocols, that encrypt network traffic either by applying special symmetric or asymmetric encryption algorithms to the payload (*e.g.* using HTTPS [9]) or by encapsulating the entire packet with an outer layer hiding the original packet headers (*e.g.* using TOR [10]). It is estimated that today half of the global network traffic is encrypted [11] and the tendency to switch from unencrypted to encrypted network protocols is rising (*cf.* Figure 2.2). Encryption can, therefore, take place on various levels of network traffic, making it very heterogeneous and difficult to analyze.

The output of each encryption algorithm creates a sequence of numbers that aims to resemble a random sequence of characters as closely as possible, since in a truly random sequence it is impossible to detect patterns. Detecting such patterns, however, is the key to determining the encryption algorithm used and ultimately to decrypting the message (*cf.* Section 2.5).

2.5 Pseudorandom Number Sequences

A sequence of numbers is called *pseudorandom* if it is generated by deterministic algorithms that simulate truly random sequences [13]. The quality of such artificially created random sequences can be determined by various measures, that try to evaluate how close

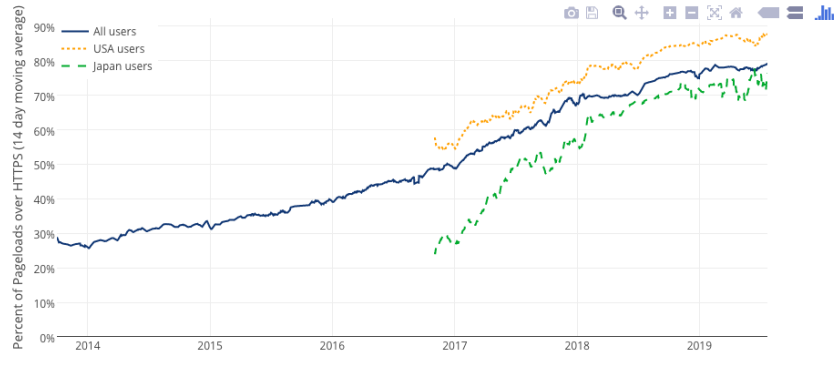


Figure 2.2: The percent of pageloads over HTTPS in the years 2014 to 2019 by different user origins [12].

a sequence of characters is to a truly random sequence. The evaluation of pseudorandom sequences helps to determine the degree and quality of encryption. The following non-exhaustive list of statistical measures are known to evaluate the quality of a pseudorandom sequence:

2.5.1 Entropy

Generally, entropy refers to the amount of disorder or uncertainty. In the context of information theory, it can be described as the information density of data [14][15] and is also known as the Shannon entropy [16]. In other words: The higher the entropy, the higher the information density of a piece of data, the lower its maximum compression rate and the closer it is to being truly random [17]. Encrypted or encoded data with high entropy indicates a pseudo-randomization that produces a sequence similar to a truly random one and therefore suggests strong encryption.

2.5.2 $\tilde{\chi}^2$ -Square Test

The $\tilde{\chi}^2$ -Square Test is a measure to evaluate the randomness of data by testing whether the difference between the expected and the observed frequency is significant [18]. Because the $\tilde{\chi}^2$ -Square Test is known for being extremely sensitive to errors in pseudorandom generators, it is the most commonly used test for this use case [19]. By calculating an absolute number from the $\tilde{\chi}^2$ -Square distribution for a stream of bytes, the test indicates how frequently a truly random sequence would exceed the calculated value.

A percentage greater than 99% or less than 1% points towards a sequence that is most certainly not random, a percentage between 99% and 95% or between 1% and 5%, points toward a sequence whose randomness is suspect, percentages between 90% and 95% and 5% and 10% indicate the sequence is “almost suspect” [17].

2.5.3 Arithmetic Mean

The simple arithmetic mean of a sequence is calculated by summing up all the elements (*e.g.* bits) of a sequence and dividing this sum by its length. Random sequences show values close to the value in the middle of the maximum and the minimum (for binary data: 0.5). Sequences that deviate from this value, are most probably non-random [17].

2.5.4 Monte Carlo Value for Pi

The Monte Carlo Method is a technique for calculating π : By taking a uniformly distributed sequence of random points x_i, y_i between 0 and 1 and a circle with radius 0.5 within the unit square (1 x 1) and by determining the ratio $p = \frac{m}{n}$ such that m is the number of points that satisfy $x_i^2 + y_i^2 \leq 1$ and n is the total amount of numbers in the sequence, a convergence towards π can be observed [20].

If a chosen sequence is truly random, π is approximated correctly. If however, the sequence is pseudo-random, its deviation from real randomness can be assessed by applying the Monte Carlo Method and comparing the calculated value with π . The greater the difference, the less random the sequence is [17].

2.5.5 Serial Correlation Coefficient

In a random sequence, a pair of successive numbers are independent of each other and uniformly distributed. By counting the number of times a pair of numbers occur in a sequence, the independence of a number based on its predecessor can be determined. This idea generalizes to higher-order n-tuples [19] as well. Based on this idea, the serial correlation coefficient measures to which extent each byte of a sequence depends upon the previous bytes, resulting in a score between 0 and 1. For random sequences, this number is close to zero. In pseudorandom sequences, however, the coefficient reaches values around 0.5 for normal pseudo-random sequences and values around 1 for highly predictable sequences such as *e.g.* uncompressed bitmaps [17].

2.6 OS-level Virtualization System

“OS-level virtualization refers to an operating system paradigm in which the kernel allows the existence of multiple isolated user-space instances, called containers.”[21] These containers are virtual operating systems, that can be run in parallel on the same host system without interfering with each other and without altering the host’s operating system [22]. OS-level virtualization has the advantage that containers can represent different applications that interact with each other without the need for distinct physical components. In addition to this, OS-level virtualization makes it possible to isolate the dependencies of the individual applications by allowing each application to have a different environment.

2.7 Web Application

A web application is an application that, unlike traditional applications, runs in a web browser [23]. Web applications are most commonly served to a client by a web server. They have several advantages over traditional applications, as they are platform-independent, easy to update and deploy, offer a similar user experience across different web apps and store their data remotely. At the same time, however, they also have several disadvantages. They have limited access to computing power and storage and are entirely dependent on the web browser, in terms of both stability and usability [23].

Using local servers either directly on the host system or by using OS-level virtualization systems, the web application paradigm can be used entirely on one single physical system without the use of external physical servers.

2.8 Related Work

2.8.1 ENT

ENT is a program developed by John Walker, that applies various statistical tests to any sequence of values (*e.g.* byte sequence for files). The program helps to evaluate the information density, compression or encryption of a byte sequence by inspecting the degree of randomness generated by the pseudorandom sequence generator (*cf.* Section 2.5) the sequence originates from [17]. To do so, it uses measures such as entropy (*cf.* Subsection 2.5.1), Chi-square Test (*cf.* Subsection 2.5.2), Serial Correlation Coefficient (*cf.* Subsection 2.5.5) and Monte Carlo Pi calculations (*cf.* Subsection 2.5.4).

2.8.2 Tranalyzer

Tranalyzer is a C-based traffic dissector (*cf.* Section 2.2) and analysis tool that includes flow generation as well as packet analysis functionalities. It is specially designed for the analysis of extremely large amounts of data, that need to be reduced to a subset, specific flows or even single packets. It includes its own text-based “packet mode” for the inspection of single packets and is implemented to work with both live and offline data [24].

2.8.3 CyberChef

CyberChef is a data analysis tool for the explorative analysis of any data format, encryption, parsing, encoding related problem. It allows inspecting any type of data input by applying a combination of manipulations in an intuitive drag-and-drop-like manner [25]. CyberChef is included in Moloch (*cf.* Subsection 2.8.6), to help with the inspection of network traffic packet contents.

2.8.4 Wireshark

Wireshark is a network traffic dissector, allowing for both live as well as ex-post analysis of network traffic. It offers a graphical user interface for exploratory in-depth analysis of different flows, sessions or single network traffic packets [26][6]. Wireshark is most suitable for all network analysis with limited data and a focus on packet contents. For inspections of larger amounts of data or inspections that focus on the broader picture of network communication, other tools such as Moloch (*cf.* Subsection 2.8.6) are more suitable.

2.8.5 Elasticsearch

“Elasticsearch is a distributed, open-source search and analytics engine for all types of data, including textual, numerical, geospatial, structured, and unstructured.” [27] Built on Apache Lucene, Elasticsearch provides a simple REST API for interaction with its database engine [27].

2.8.6 Moloch

Moloch is a “large scale, open-source, indexed packet capture and search system” [28]. It consists of a *capture* component, a C-based application that handles the monitoring of live network traffic or the parsing of offline network traffic, and a *viewer* component, that offers a session-based web interface to the data. Moloch uses Elasticsearch (*cf.* Subsection 2.8.5) as the underlying database. During the capturing of network data, all metadata extracted by Moloch is therefore written to an Elasticsearch instance running in the background. For a more in-depth investigation of network data, Moloch offers direct access to the packet contents connected to each session.

2.8.7 Docker

Docker is a software solution offering OS-level virtualization (*cf.* Section 2.6). It consists of the docker daemon, which enables the host system to run different containers, and the docker hub, which allows sharing pre-built containers [29].

“A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, run time, system tools, system libraries and settings.” [30] The container “isolates software from its environment and ensures that it works uniformly despite differences for instance between development and staging [30]” through the docker daemon, which provides the interface between a container and the operating host system.

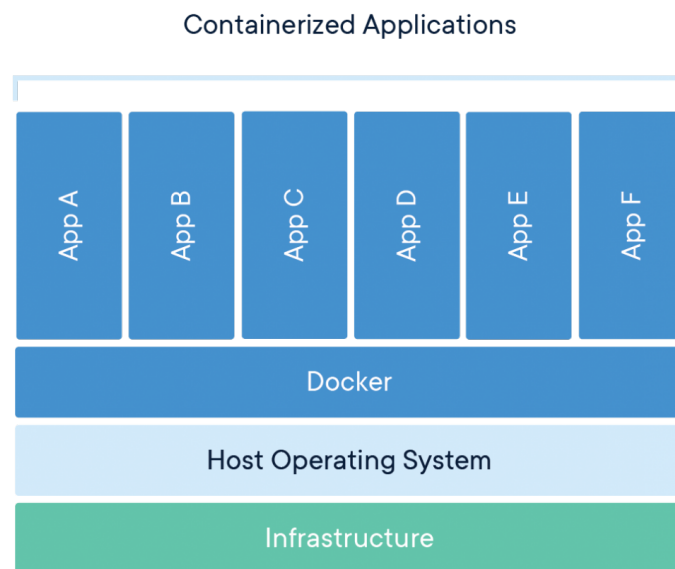


Figure 2.3: An abstract view of the functionality provided by the docker environment through the docker daemon and the different containers (here A - F) that hold different applications [30].

Chapter 3

Architecture

This chapter describes the application developed within the scope of this project at a high level, including elaborations of high-level design considerations (Section 3.2), requirements (Section 3.3) – split into project goals (Subsection 3.3.1, quality requirements (Subsection 3.3.4) and functional requirements (Subsection 3.3.3) – and assumptions (Section 3.4) made. Further, it includes a specification of the systems general topology (Section 3.5).

3.1 Naming

The choice of the name for the application, *Kronos*, was inspired by the application *Moloch* (*cf.* Subsection 2.8.6), in use at KAPO ZH today. *Moloch*, related to the biblical Hebrew word for “King”, is the biblical name of a “Canaanite god associated with child sacrifice, through fire or war” [31]. It is then through this rather somber connection of child sacrifice that *Moloch* has been associated with the Carthaginian god Baal Hammon, an alternative name of which is Kronos (also Cronus or Cronos). It is due to this connection and ignoring the negative connotations that it was found to be a fitting name for our application. The system developed will, therefore, henceforth be referred to as *Kronos*.

3.2 Design Considerations

Since there are various tools available to investigate and inspect network traffic data (*cf.* Sections 2.8.6, 2.8.4 and 2.8.2), the designed system needs to offer substantially different features and abilities, than those provided by other existing systems. Additionally, while the system applies to any situation, where the basic assumptions (*cf.* Section 3.4) are met and where network traffic data needs to be inspected, it is developed specifically for the analysis and inspection of large amounts of surveillance data. The following section describes the project goals, the use cases and the specific requirements arising from this scenario.

3.3 Requirements

3.3.1 Project Goals

In the aforementioned context of the analysis and inspection of surveillance data, the application needs to replace automatable, simple and repetitive tasks, while on the other hand offering valuable insights for investigation tasks. The following concrete use cases were determined through meetings with stakeholders at KAPO ZH.

1. **Insights into applications or devices in use:** In network forensics, it is very important to know as much about the devices in place as possible. Such information can be helpful to coordinate future investigations, for the analysis of seized devices or to gather clues about the bigger picture, such as *e.g.* the absolute amount of devices in place. Information about the applications that generate network traffic is also crucial. Depending on the specific scenario, certain traffic destination IPs might carry useful information about this. The ability to filter the data by looking *e.g.* at traffic to/from a specific host during a specific period is, therefore, of prime importance.
2. **Automated identification of unencrypted network traffic:** While the headers may indicate whether a packet's contents are encrypted, this is not always the case. Communication may be encrypted on the application layer even though the headers indicate otherwise. Conversely, packet contents may be unencrypted even though the headers indicate otherwise if there were errors along the way or if the protocol was not implemented correctly. The second case, especially, is very interesting in network forensics since sensible information may be revealed that was thought to be confidential.
3. **Identify patterns or anomalies:** The usage patterns found within surveillance data, may provide clues about the amount of data a device sent, its destination, the time of the day it was transmitted and, finally, reveal patterns about the user's activity. Additionally, assumptions on the daily routine of a person of interest can be made based on the device and traffic usage patterns. For example, the presence or absence of network traffic originating from a specific device during a specific period may indicate whether a person of interest is physically present in a location or not.

In addition to this, the following more latent goals regarding quality were formulated:

4. **Maintainability:** The code should be easy to maintain, improve and fix and should be well-documented. It further should be configurable to adapt to the specific data in place.
5. **Usability:** The application should be easy and intuitive to use.

Based on these project goals and within the specific context of the KAPO ZH (*cf.* Subsection 3.3.2), the functional and quality requirements described in Subsections 3.3.3 and 3.3.4 were derived.

3.3.2 Digital Forensics at KAPO ZH

In the years between 2014 to 2018, the canton of Zürich executed over 3000 real-time monitoring jobs, a score higher than most other cantons in Switzerland (*cf.* Figure 3.1). The reasons for the real-time monitoring jobs are among others: drug-related crimes, offenses against property and offenses against life and limb (*cf.* Figure 3.1). Even though the numbers for such assignments are decreasing, the monitoring of suspect activities through the surveillance of network traffic remains one of the main tasks for – not only the federal but also – the cantonal police of Zurich (*cf.* Figure 3.3).

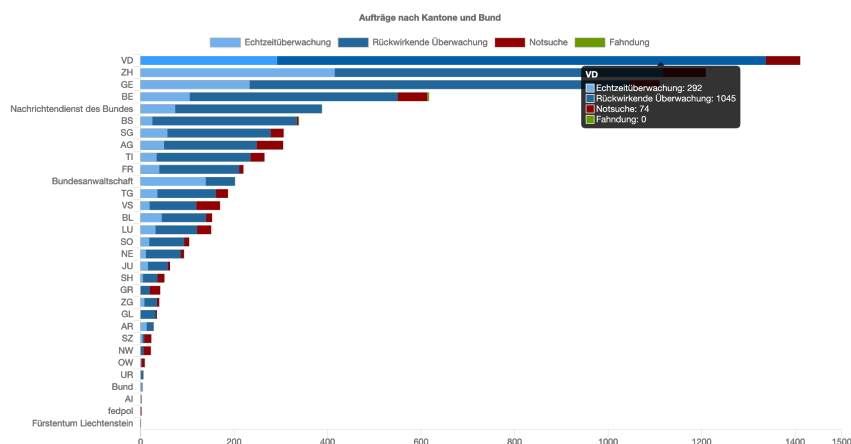


Figure 3.1: A comparison from 2018 between all Swiss cantons regarding their surveillance activities [32].

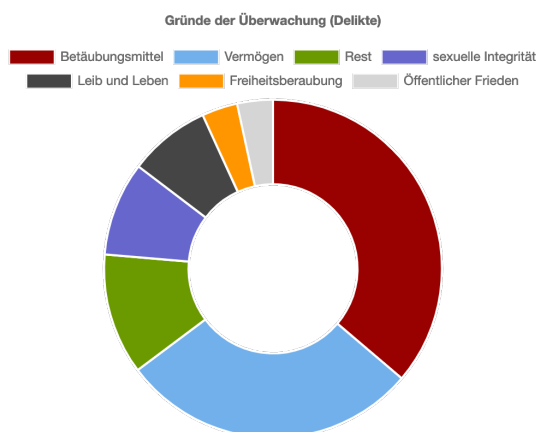


Figure 3.2: Reasons for surveillance for the canton of Zurich in 2018 [32].

The digital forensics group at KAPO ZH is responsible for recovering, analyzing and securing data potentially connected to criminal activities. Among others, the analysis of surveillance data, especially of network traffic data, is one of their areas of expertise. Their typical work consists of securing and recovering data from mobile devices or computers as well as the ex-post analysis of network traffic data.

KAPO ZH relies mainly on the Moloch-Elasticsearch combination (*cf.* Subsections 2.8.6 & 2.8.5) as tooling for their network forensic tasks. Apart from this, analysis is sometimes

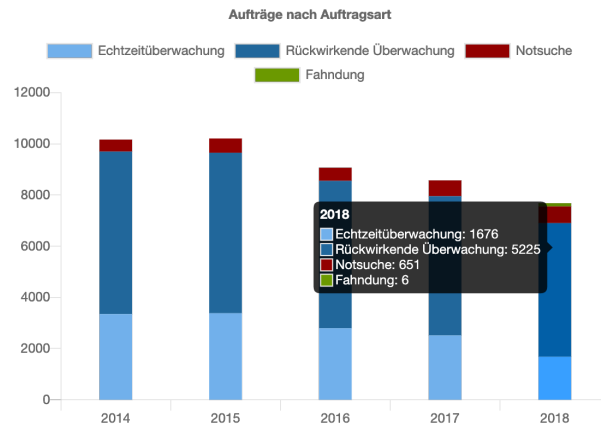


Figure 3.3: Comparison of different kinds of surveillance jobs issued in Switzerland in the years 2014 to 2018 [32].

performed using Bash or Python scripts. This is mostly the case if operations are not supported or if they would be too cumbersome to accomplish within Moloch.

3.3.3 Functional Requirements

The functional requirements describe all feature related requirements that were derived from the use cases described above.

1. **Visualization of DNS requests for hostname over time:** To reveal usage patterns for different applications, *Kronos* should visualize the appearance of DNS requests for hostnames related to the application over time. A frequent appearance of DNS requests may indicate active use of the application in question.
2. **Visualization of the amount of transmitted data for hostname over time:** To reveal the type of use for different applications, *Kronos* should visualize the amount of data transmitted to and from hostnames related to the application over time. This may provide additional information about the mode in which the application was used (*e.g.* voice call or chat for messenger applications).
3. **Visualization of user agent appearance over time:** To reveal information about the type of devices in use, *Kronos* should visualize the appearance of different user agents over time. This may reveal information about the number of devices in use, the operating system versions installed, as well as the presence or absence of a device in the network.
4. **Display of aggregated user agent information:** To provide a quick overview of the devices in use, *Kronos* should display a summary of the user agents present in the traffic data.
5. **Automated detection of (un-)encrypted traffic:** To reveal readable parts of the traffic data with potentially useful information, *Kronos* should automatically detect encrypted or unencrypted packets, even if the headers indicate otherwise.

3.3.4 Quality Requirements

The quality requirements include all properties that are related to the usage, life-cycle and maintenance of the application.

1. **Usability:** The application needs to be easy to use and to configure. Its user interface has to provide laypersons and professionals with simple and understandable options for the investigation of the network surveillance data.
2. **Maintainability:** The application's source code needs to be easy to maintain.
3. **Future developments:** It should be possible to easily add new features to the application.

3.4 Assumptions

Based on the specific scenario at KAPO ZH (Section 3.3.2) and the aforementioned general considerations (Section 3.2), several assumptions can be made:

1. **Existing Environment:** The application may rely on a running ensemble of a Moloch instance (*cf.* Subsection 2.8.6) and the corresponding Elasticsearch instance (*cf.* Subsection 2.8.5).
2. **Accessibility of Data:** The data, parsed by Moloch and stored via Moloch in Elasticsearch, is retrievable from Elasticsearch without any restrictions other than those made by the Elasticsearch API itself (*e.g.* no password restrictions on Elasticsearch).
3. **Security of Connections:** It is assumed, that all connections between the application and its data sources (Elasticsearch and Moloch) happen via a secured internal network and do not need to be secured further.

3.5 System Topology

To fulfill the requirements of a usable and appealing user interface and highly performant data processing, as well as enabling efficient interfaces with the existing infrastructure, the system was designed to fit a basic client-server architecture. Figure 3.4 shows an abstract view of the system components included in the *Kronos* system.

The client is responsible for any interaction with the user, including the handling of the login procedure, the initialization of data retrieval and the visualization of the forensic network data in an appealing way. To minimize dependencies in the run time environment, as well as facilitate the sharing of visualizations and data between individuals, it is designed to run in a web environment. The client queries the server for any data that it

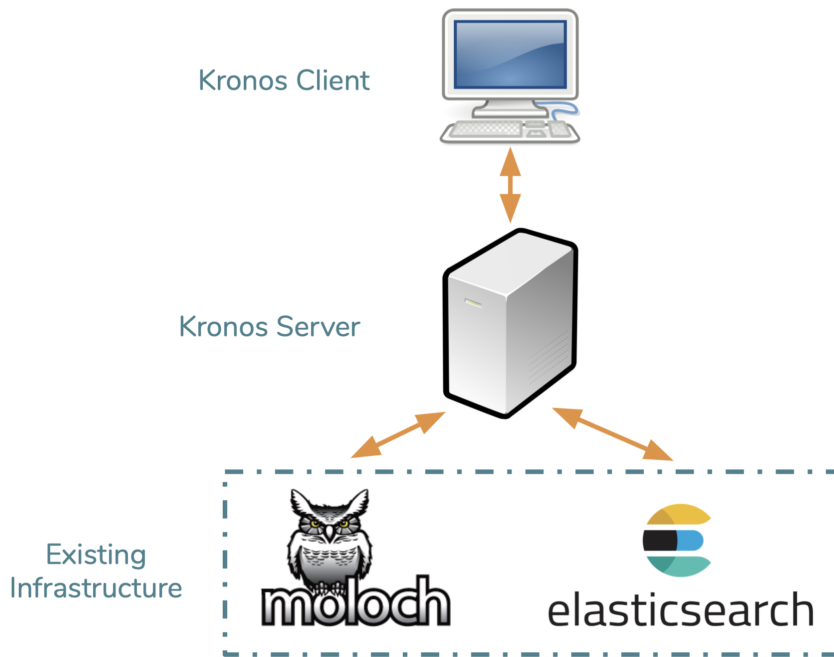


Figure 3.4: An abstract view of the *Kronos* system topology

needs and processes this data only as much as it is necessary to visualize it. The brunt of the data processing is offloaded to the server, since the client is limited in optimizations, such as parallelization, due to the web-based environment. The client also reacts as a slave to any configuration changes in the server.

The server is designed to run as a standalone application separate from the client. In essence, it is responsible for interfacing with the existing infrastructure, retrieving the necessary data from it and serving it to the client in a *Kronos*-specific format. This may involve performing transformations and aggregations on the data. To minimize the processing load, however, the server is kept lightweight and delegates as many of the resource-intensive operations to the existing infrastructure as possible. This allows the application to scale to large amounts of data. Performance is helped further by handling client requests in parallel and parallelizing queries to the existing infrastructure. To fulfill the requirement of maintainability and extensibility, the server's basic functionality is designed to be configurable, such that it can be extended to new types of network traffic without having to modify the source code. The server is multi-tenant capable and able to communicate with multiple Moloch-Elasticsearch ensembles at the same time. Figure 3.5 shows a setup that has the server connected to multiple clients and communicating with multiple Moloch-Elasticsearch ensembles simultaneously.

The final part of the *Kronos* system topology consists of the externally running services used by the application. Mainly, *Kronos* requires the following four components:

1. Authentication and Authorization Provider
2. PCAP Parser
3. Fast Traffic Metadata Search Engine

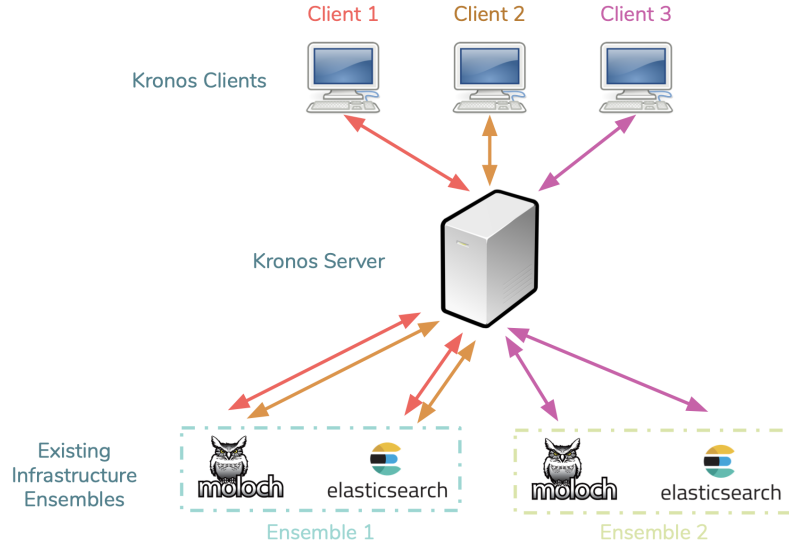


Figure 3.5: An abstract view of a *Kronos* setup with multiple tenants and multiple Moloch-Elasticsearch ensembles. The data flow for each client is shown in a different color.

4. PCAP File Repository

In the context at hand, these tasks are fulfilled by the existing infrastructure in use at KAPO ZH today. As is described in Subsection 3.3.2, this consists mainly of the ensemble of Moloch and Elasticsearch. It has the advantage of being highly scalable and of performing a variety of tasks related to the extraction of metadata from network traffic data, the grouping of individual packets into sessions and the storage of the relevant traffic data and metadata in an indexed manner, such that it can be accessed efficiently. The *Kronos* system, therefore, aims to profit from these advantages as much as possible. Within this context of the *Kronos* system, the existing components are used as follows:

Elasticsearch is used as a fast search engine for traffic metadata (3). This ensures that access can be achieved in an efficient and performant way. It also avoids *Kronos* having to maintain a separate database and the consistency issue that would accompany this.

Moloch is used mainly in two ways. First of all, Moloch maintains accounts with specific access rights to certain subsets of the data and is therefore responsible for authentication and authorization (1). *Kronos* thus delegates the account maintenance, as well as the handling of the login verification to Moloch. Second, it parses raw PCAP files (2), extracts metadata, loads it into Elasticsearch and stores the PCAP files after some pre-processing. Any data in *Kronos* is therefore parsed and pre-processed through Moloch, which avoids having to re-implement the extraction infrastructure in the *Kronos* server. If in some cases, *Kronos* needs access to parts of the data that are not available through Elasticsearch, the corresponding PCAP files can also be obtained from Moloch, which serves as the PCAP file repository (4).

Chapter 4

Implementation

The following chapter explains the implementation of the system outlined in Section 3.5. Apart from the requirements described in Subsections 3.3.3 and 3.3.4, development was guided by the following principles:

- Well-established software engineering practices should be followed
- Modern languages and frameworks should be used
- A modern development process should be implemented

4.1 Kronos Client

4.1.1 Languages and Frameworks

Typescript

TypesScript is a superset of JavaScript that includes (optional) static types [33]. This enables more sophisticated developer tools compared to regular JavaScript and produces more comprehensible code. It is transpiled to regular JavaScript and, therefore, integrates well with existing code and libraries. Due to these advantages, TypeScript is used as a basis for Angular development, a popular web application framework. Typescript is developed and maintained by Microsoft.

Angular

Angular is an open-source web application framework developed by Google [34]. While the original version of Angular, AngularJS, was based on JavaScript, current versions use TypeScript as their main programming language. Angular includes a variety of features for the development of web applications, such as facilities for routing, an HTTP

client, a dependency injection system and mechanisms for binding HTML templates to a TypeScript class.

In its basic form, an Angular application consists of a hierarchy of Angular components. Each Angular component corresponds to a TypeScript class that defines the application logic and that is tied to an HTML template definition, and may also include its own CSS styling. Furthermore, an application may include classes that can be provided globally through the dependency injection system, called services. Data is transferred between components using a one-way property binding, meaning that component properties can be bound to components further down in the component hierarchy. To send data upward in the component hierarchy, events are generated that can be subscribed to in an upstream component. Each component has a lifecycle that the developer can react to using lifecycle hooks, which exist, for example, for the initialization and the destruction of a component.

Angular also includes a developer tool, called the Angular-CLI, that streamlines many common operations, such as the generation of new components, the initialization of a project, or building and packaging the application for development or production. It also manages and automatically retrieves dependencies through the Node Package Manager (NPM).

Angular is used across the industry and remains one of the most popular web development frameworks to this date [35]. It brings along a rich ecosystem of third-party components and libraries that allows for fast and effective development of applications.

Angular Material

Angular Material is an open-source web component library for Angular developed by the Angular development team at Google [36]. It follows Google’s material design guidelines to provide a modern and intuitive web experience and includes features for internationalization and accessibility. Components range a variety of interaction patterns, including elements for layouting, form controls, menus, data tables, as well as dialogs and popups.

ngx-charts

ngx-charts is an open-source declarative charting framework for Angular built on top of D3js [37]. ngx-charts integrates tightly with Angular and uses its property and event binding mechanisms internally. The resulting charts are composed of SVG elements and also include extended functionality, such as tooltips or the generation of events for selections of individual chart elements. ngx-charts provides a library of different types of charts including line, bar and area charts. It is also extendable to allow for the creation of custom charts using their existing framework.

4.1.2 Design Patterns

The architectural patterns implemented in the *Kronos* client application are dictated primarily by the Angular framework and its best practices. Consequently, the application

is constructed with a hierarchy of components as its primary architectural characteristic. In its core, Angular follows a component-based Model-View-Controller pattern [38]. Each component consists of a view, represented here by an HTML template file, which is controlled by the component class. The component class – the controller – in turn, acts as an interface between the view and the data model.

Also following the Angular framework, dependency injection is used to provide instances of component or service classes. This ensures, for example, that these classes can be exchanged for testing purposes or dependent on the runtime environment. Figure 4.1 shows an overview of how the building blocks of an Application are related.

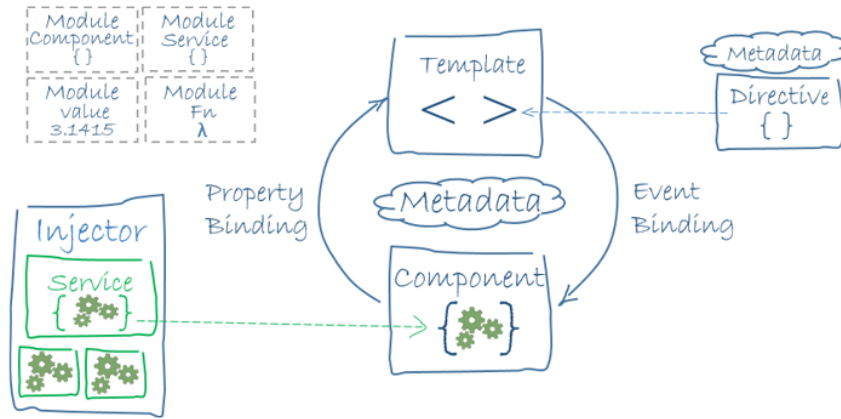


Figure 4.1: An overview of the building blocks of an Angular application [34]

4.1.3 Implementation Details

The *Kronos* client application was implemented fully as an Angular web application. As a consequence, Typescript was used as the main programming language. The build tools from the Angular-CLI were used extensively, including dependency management with NPM and packaging with Webpack.

The main implementation effort for the *Kronos* client application went into creating customized charts for the network traffic point statistics of interest. The Angular charting library, *ngx-charts*, was used as a starting point for these visualizations. New charts that fit the very specific use cases at hand were then created based on existing components from *ngx-charts* and their facilities for the creation of custom charts. To increase the performance of the charts, some of the property and event binding mechanisms implemented in *ngx-charts* were removed. This was necessary to deal with the large amounts of data in the *Kronos* application.

This effort resulted mainly in two custom visualizations: The timeline chart visualizes time intervals for multiple different classes in a common time domain (*cf. e.g.* Figure 5.7). A subrange in the time domain can be selected to allow for closer inspection of the intervals in said subrange. The timeline multi-bar chart visualizes the development of a variable over time in a histogram for multiple different classes in a common time domain

(cf. e.g. Figure 5.15). Similar to the timeline chart, the time domain can be filtered for closer inspection.

The rest of the user interface was implemented using the Angular Material web component library. This allowed us to efficiently create a user experience that is responsive, fast and visually appealing. The client was built to act as a slave to the server configuration. In practice, this means that UI components are added or removed dynamically based on the responses received from the server.

4.2 Kronos Server

4.2.1 Languages and Frameworks

Kotlin

Kotlin is an open-source statically typed programming language released by Integrated Development Environment (IDE) developer JetBrains in 2016 [39]. It was originally developed to provide features that were found to be missing from other programming languages by the JetBrains developers. Kotlin has since gained wide popularity across the industry. Most notably, Google has adopted Kotlin as the preferred programming language for Android in May of 2019 [40].

Kotlin can be compiled to JavaScript code, to run natively or to run on the Java Virtual Machine (JVM). In its JVM version, it is fully interoperable with Java. This means that existing Java frameworks and libraries can be used seamlessly in applications developed using Kotlin. Together with the implementation of innovative language features, such as nullable types or data classes, that lead to higher expressiveness of the language, and the support of asynchronous programming at the language level, this makes it an ideal choice for the development of server-side applications [41].

Kotlin approaches asynchronous programming through the concept of Coroutines [42]. In essence, Coroutines are light-weight threads. They differ, however, by not being tied to a single thread. Instead, a Coroutine can run on any available thread from a pool and, consequently, a much higher number of Coroutines may exist than would be possible for threads. Apart from asynchronism, Coroutines can also be used for several other constructs, such as concurrency.

Ktor

Ktor is an open-source framework for asynchronous connected applications developed by JetBrains, the company behind Kotlin, and built using Kotlin and Kotlin Coroutines [43]. It provides facilities for both server-side and client-side communication and includes features, such as authentication, session management and cross-origin resource sharing (CORS). Currently, Ktor only works with the Kotlin version for JVM. The goal in the

long run, however, is to implement a version that does not rely on external dependencies and, therefore, works with any Kotlin flavor.

Koin

Koin is an open-source lightweight dependency injection framework for Kotlin [44]. Other than more heavyset frameworks, Koin forgoes the usage of reflection or code generation, relying instead on a pure functional resolution, which is easily achievable in Kotlin. Koin integrates with Ktor and can be started easily through the available Ktor lifecycle hooks.

Elasticsearch Java High Level REST Client

To access Elasticsearch APIs from a Java application, Elasticsearch provides several Java libraries [45]. The Elasticsearch Java High-Level REST Client is built on top of the more minimal Elasticsearch Java Low-Level REST Client. It provides methods for a wide range of Elasticsearch functionality, including search, aggregation and access to documents and indices. All methods can be called synchronously and asynchronously.

4.2.2 Design Patterns

At the most abstract level, the *Kronos* server application is built to fit a hexagonal architecture. This architectural pattern aims to ensure a clear modularization of the application, separating the domain logic from any logic necessary to access external resources [46]. To this end, the application is separated into three modules: the domain, the ports and the adapters. The domain contains any core business logic, whereas all external dependencies, such as the client, repositories, etc., are accessed through adapters. The ports act as the interface between the domain and the adapters. They can again be divided into ports *provided* by the domain and ports *required* by the domain. Figure 4.2 shows an abstract view of the hexagonal architecture applied in the context of the *Kronos* server application. The key takeaways here are:

- The domain does not depend on the concrete implementation of the communication with the client, Moloch or Elasticsearch
- It is irrelevant for the domain whether data is retrieved from Moloch or Elasticsearch

The *Kronos* server application follows the dependency injection (DI) design pattern. The DI pattern is a creational pattern that implements the inversion of control principle [47]. Hereby, individual components retrieve their dependencies from a dependency injector instead of instantiating them themselves. Since dependencies are mostly bound via interfaces, this enables dependency injection frameworks to exchange concrete implementations at runtime, e.g. depending on the runtime environment. In addition to this, the dependency injection pattern promotes loose coupling, ensuring proper modularization of the application and improving testability.

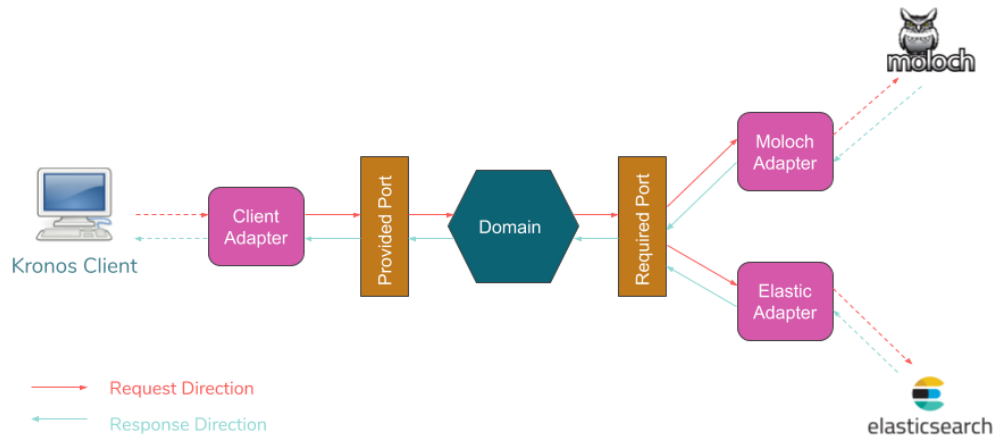


Figure 4.2: An abstract view of the *Kronos* server application's hexagonal architecture. Data flow in request and response direction is shown in red and blue respectively.

Finally, the *Kronos* server application is designed to be configurable. A single configuration file dictates large parts of the information offered by *Kronos* to the user. The server application, therefore, reacts to the current configuration, which is loaded at startup. This ensures easy adaptations for basic changes without having to touch the source code.

4.2.3 Implementation Details

The *Kronos* server application was built entirely using Kotlin JVM to run on the HotSpot JVM included in OpenJDK8. This was a necessity to use Ktor which currently only works with the JVM version of Kotlin. *Kronos* server runs as a standalone embedded server application, meaning that there are no further requirements, such as a servlet container, that need to be fulfilled to run it other than the OpenJDK8 Java Runtime Environment (JRE) being installed.

The separation of modules in the hexagonal architecture mentioned above was implemented using Gradle modules. This allows us to separate the architectural components clearly and to ensure that dependencies between the domain and the adapters can only be accessed through a port interface.

The application uses Ktor both for handling requests from the client using the Ktor server module, as well as for launching requests to Moloch using the Ktor client module. Development included developing a fix for the HTTP digest authentication built into Ktor for it to comply with the RFC specification, which will be included in Ktor release 1.3 [48].

To improve the performance of the *Kronos* server application and to profit from the scalability and efficiency of the existing infrastructure, the decision was made to move

as many of the data-centric operations outside of our application to be performed on the existing Elasticsearch instance. In practice, this meant moving them outside of our domain module and into the Elasticsearch adapter.

Kotlin coroutines are used for concurrency to improve the speed of our application further. Parallelization happens at the request level, meaning that both incoming client requests are handled concurrently and outgoing requests to Elasticsearch or Moloch are performed concurrently.

4.3 Processes

The following section will outline the core processes in the *Kronos* application, including the interactions between the components involved. Each process will be outlined in a sequence diagram.

4.3.1 Login

The login process is kicked off by the user requesting an action from the client that requires a valid Moloch login, *i.e.* the loading of data from Elasticsearch or the generation of a Moloch link. The client first checks with the server whether a session is stored for the user performing the request. If so, the session is checked for its validity by performing a request to Moloch with the credentials stored in the session. In case the credentials are valid, there is nothing else to do and the login success is communicated to the user.

In case the credentials stored are invalid, new credentials have to be obtained from the user via the web UI. These credentials are then transmitted to the server who in turn performs a request to Moloch to check if they are accepted. In case they are, the server asks Elasticsearch for any restrictions on the data that can be viewed using the credentials entered. These restrictions are returned and stored together with the credentials in a session on the server. The login success can then be communicated to the user. In case the credentials entered are invalid, the user is asked to enter new credentials via the web UI. This is repeated until the user enters valid credentials. The login process is summarized in Figure 4.3.

4.3.2 Data Retrieval

The data retrieval process is started by the user clicking a refresh button in the client UI. It is then performed in two distinct requests which will be explained separately: The communication data retrieval and the user agent data retrieval.

To obtain the communication data, the client sends a request to the server. Using the session stored for the user and the regular expressions for hostnames stored in the application's configuration file (*cf.* Figure 5.8), the server then launches a series of requests to

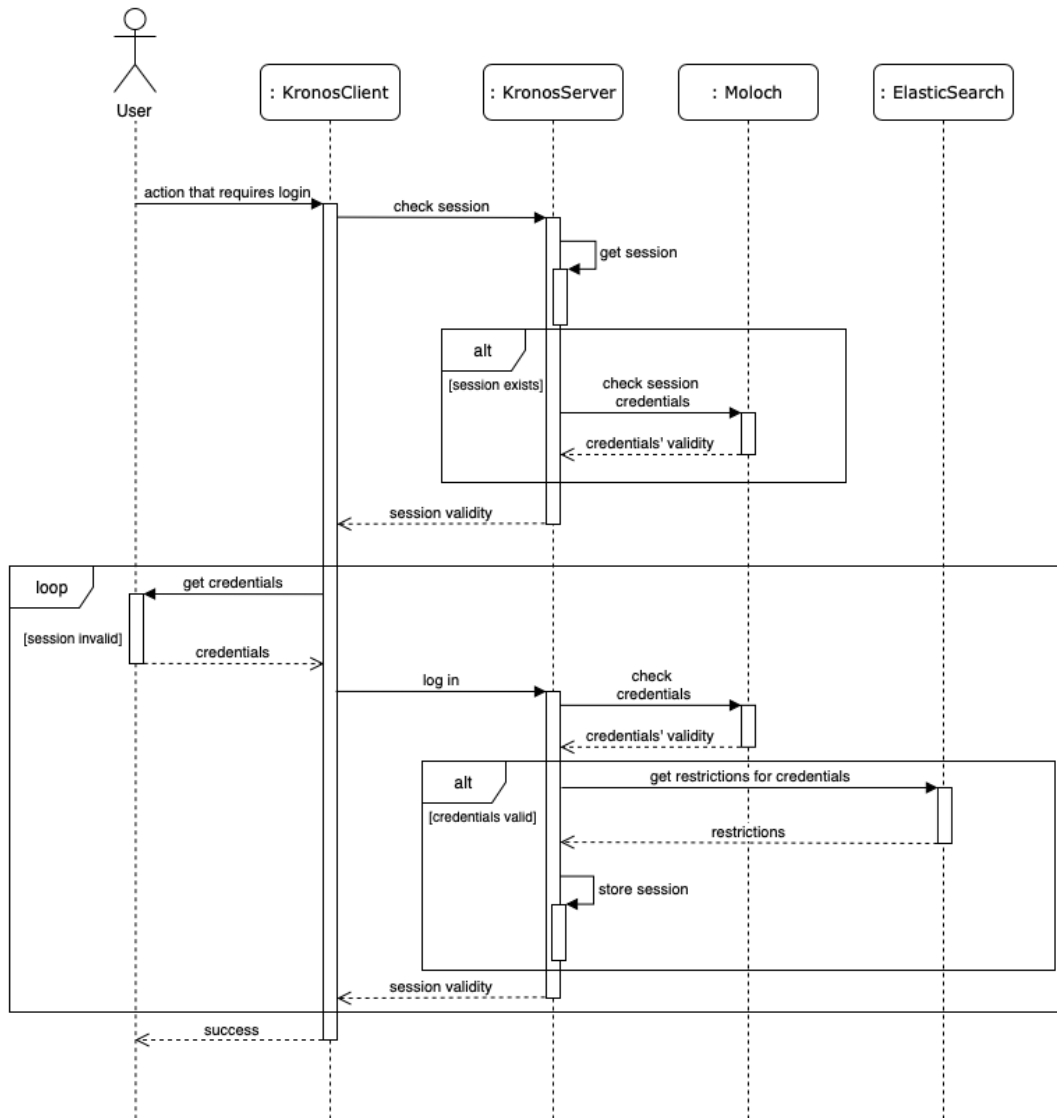


Figure 4.3: Sequence diagram of the *Kronos* login process and session management

Elasticsearch. For each regular expression, the server first retrieves all DNS requests for any hostname represented by the regular expression. This forms the first part of the communication data. The server then queries Elasticsearch for any IPs that were returned in DNS responses for any hostname represented by the regular expression. The IPs returned are then used to query Elasticsearch for an aggregation of the number of bytes transmitted from and to said IPs in a timeline. Once these three requests have been performed for all configured regular expressions, the communication data is transmitted back to the client and the successful retrieval is communicated to the user via the UI. The communication data retrieval process is outlined in Figure 4.4.

To obtain the user agent data, the client again sends a request to the server. Similar to the communication data process, the server uses the stored session and the regular expressions for user agents configured in the configuration file. For each one of these regular expressions, the server then first queries Elasticsearch for any user agent matching the

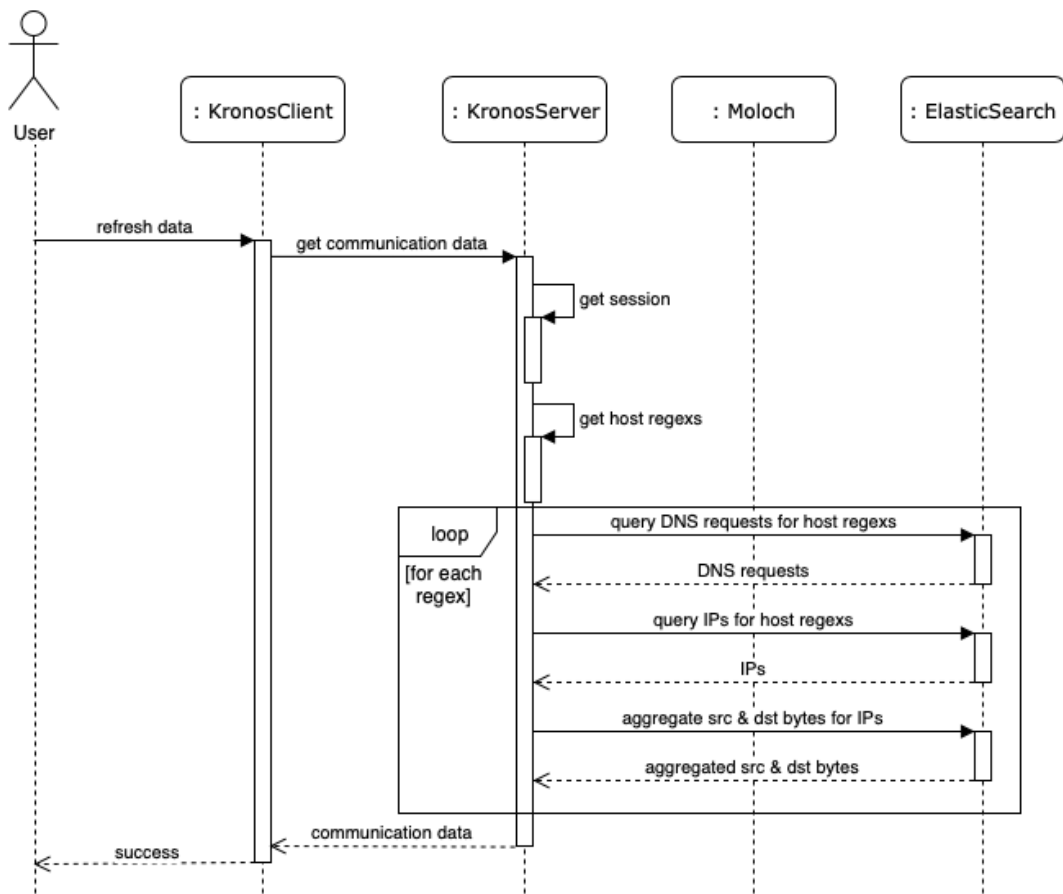


Figure 4.4: Sequence diagram of the *Kronos* communication data retrieval process

expression. A second request is launched to aggregate the appearance of these user agents into time interval buckets. The server then retrieves aggregated information on the user agents overall, including the distinct user agents, as well as the number of appearances. The user agent names are then parsed on the server to extract further information, such as OS versions or devices used. Finally, the server queries Elasticsearch for “server bag” user agents. They are used in Apple devices and provide valuable information on the device and the OS version used. This data is again aggregated in a timeline via Elasticsearch. The user agent data is then transmitted back to the client and the successful retrieval is communicated to the user via the UI. The user agent data retrieval process is outlined in Figure 4.5.

4.3.3 Link Generation

Kronos can generate direct links to Moloch for any communication data displayed in the client. To start the link generation process, the client clicks a time interval for a certain hostname in a graph displayed in the UI. The client then sends a request to the server that contains said hostname. The server obtains the Moloch host in use from the session, stored for the current user. It then queries Elasticsearch for all IPs that were returned in a DNS response for the hostname transmitted by the client. Using the Moloch host and

the IPs returned by Elasticsearch, it can then generate a link. This link is returned to the client. The user is then automatically forwarded to the Moloch link returned. The link generation process is outlined in Figure 4.6

4.4 Productive Environment and Deployment

The choice and design of the productive environment were guided by a single goal: to enable easy and fast deployment with minimal requirements on the target system. The final version of the *Kronos* application can be deployed on any machine that meets the following requirements:

- Must have Docker & Docker-Compose installed
- Must have Git installed
- Must be connected to the internet

Building and starting the application then only requires two steps:

1. Clone the *Kronos* Git repository
2. Run the Build-And-Run script

To achieve this, both the build environment, as well as the productive environment are virtualized using Docker: The client build relies on NPM to retrieve dependencies and to perform the actual compilation and packaging. An installation of NodeJs is necessary for it to work. It is, therefore, performed in a Node Docker container. The server build similarly relies on Gradle and JDK8. It is therefore performed in a Gradle-JDK8 Docker container. These build containers are only necessary for running the build script and are stopped and deleted after use.

The deployment of the build artifacts is then handled by Docker-Compose. Docker-Compose is a tool for running multi-container Docker applications [49]. The user can then build and start the Docker containers involved with a single command. Networking between the containers can also be configured via Docker-Compose. For the *Kronos* application, we start two distinct Docker containers. The *Kronos* server application simply runs in an OpenJDK8 JRE container. The container for the client application is responsible for serving the web content to the user. To this end, the container runs an NGINX web server. It also forwards any calls to the server API to the server docker container using a reverse proxy via a Docker-internal network. The only entry point to the application is thus via the webserver container. The *Kronos* Docker setup is summarized in Figure 4.7.

4.5 Development Process

To deal with the vague requirements and the uncertainty due to the existing environment at KAPO ZH, we chose an agile development process in the implementation of this project. In its origin, the Agile Manifesto [50], agile software development is outlined as follows:

- Value individuals and interactions over processes and tools
- Value working software over comprehensive documentation
- Value customer collaboration over contract negotiation
- Value responding to change over following a plan

Applied to our project, this meant that we chose to communicate continuously with our stakeholders at KAPO ZH, collected feedback as soon as features were finished and iteratively improved our application instead of specifying everything beforehand. To enable this continuous communication, as well as to have constant access to real-life data, we chose to work on-premise at KAPO ZH in Zürich.

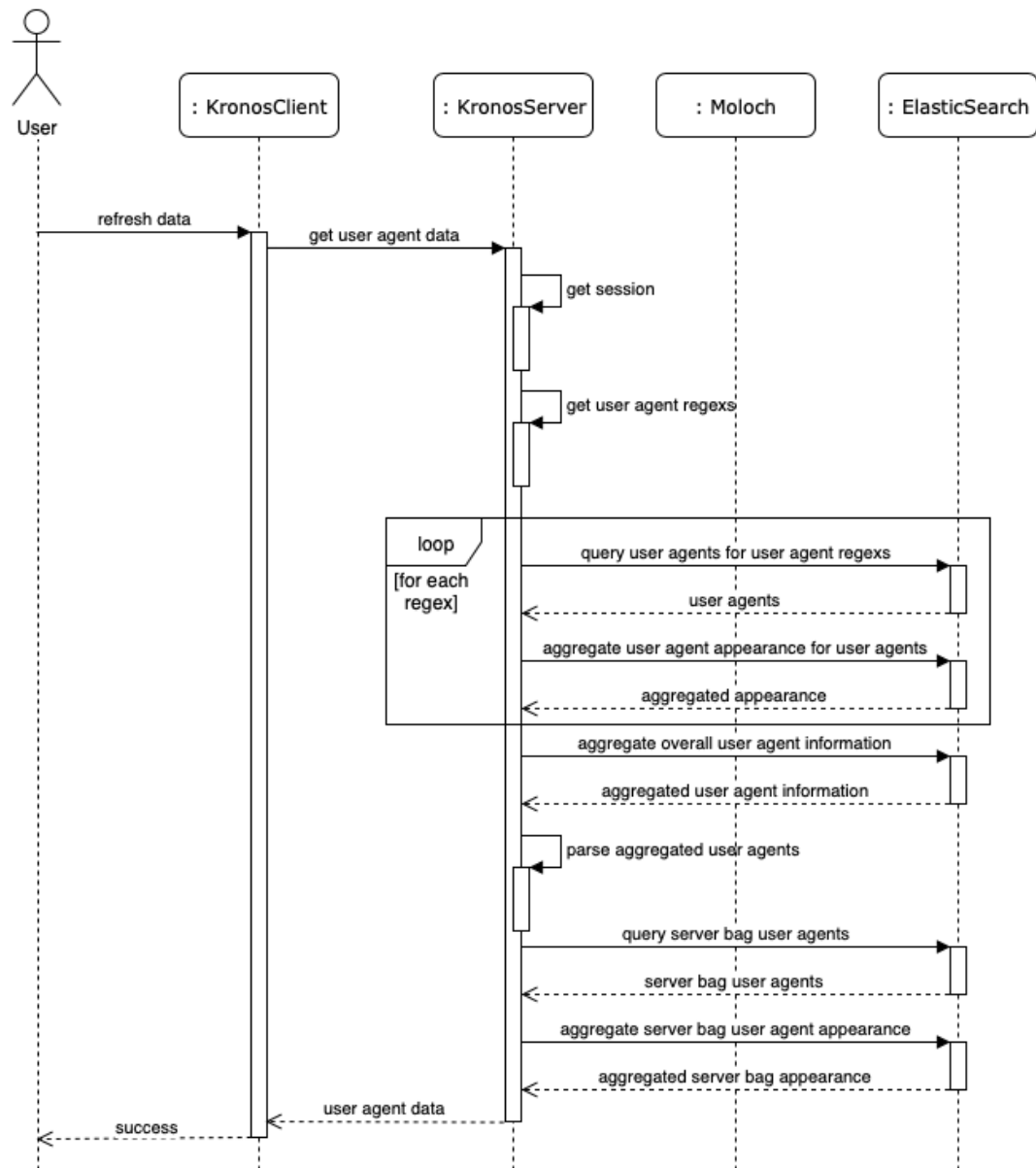
We managed our project through Github. Tasks were filed as Github Issues and managed through a Github Project board. This allowed us to keep track of our work backlog, of tasks that were in progress and of bugs we may have found along the way. Github was also used as the source code repository and version management tool. We included mandatory reviews for any code that would be merged into the application core for quality assurance.

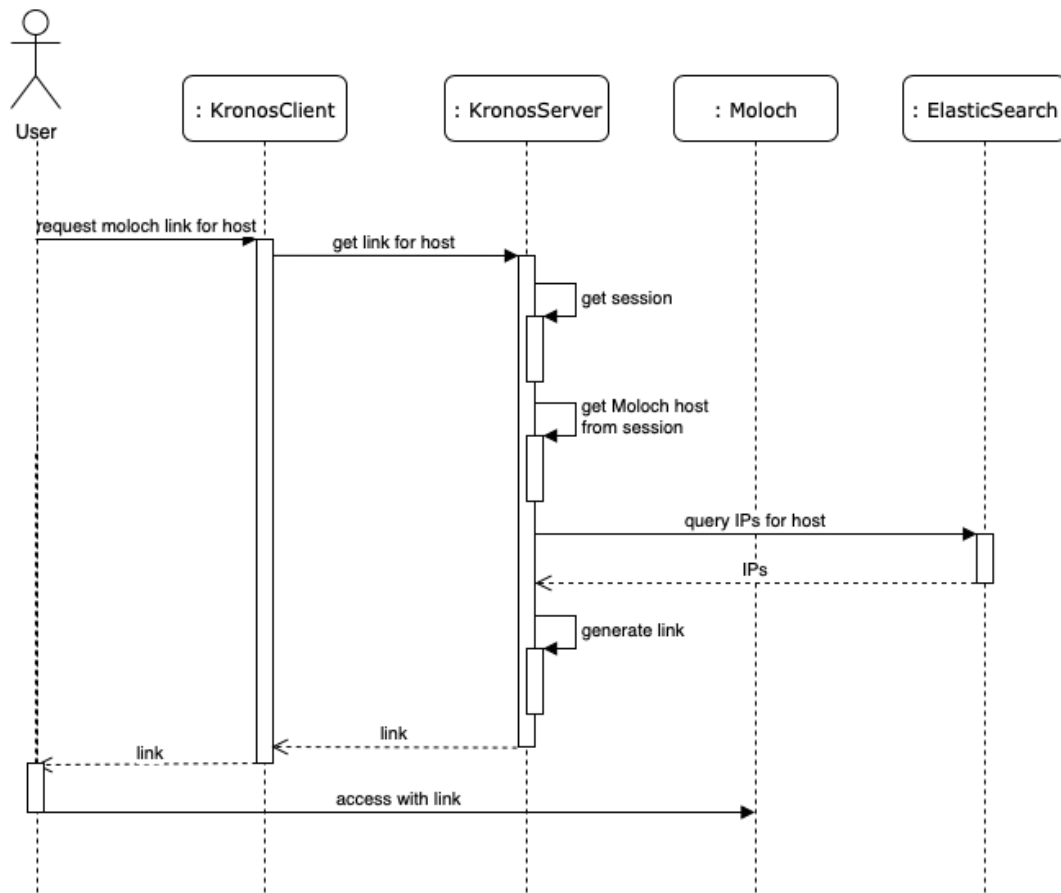
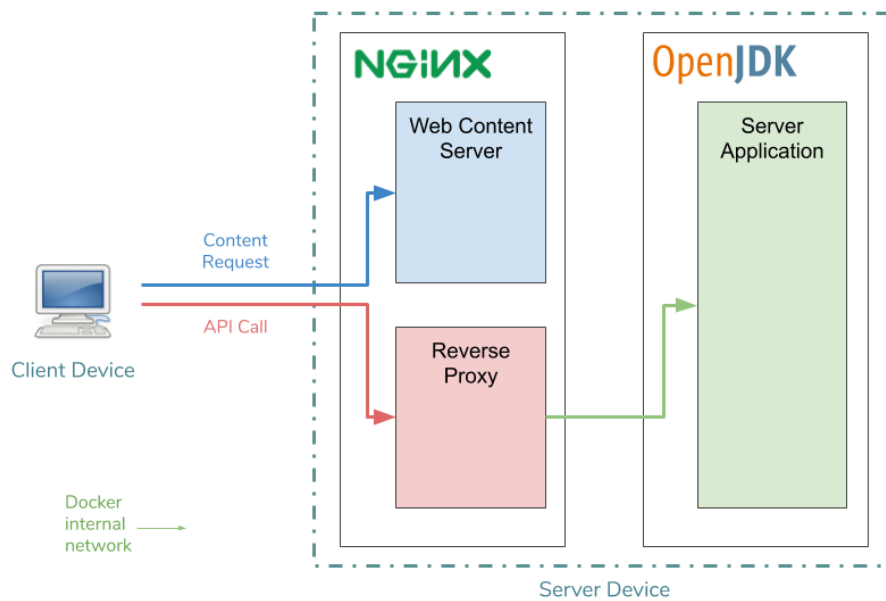
4.6 Research Into Encryption Detection

A common problem in the context of network forensics nowadays is the presence of network traffic protocols that use encryption to hide the content of the traffic they transmit (*cf.* Section 2.4). Two specific use cases, however, are specifically interesting to look at: First, payloads may not be labeled correctly by their corresponding headers, such that *e.g.* an encrypted payload is labeled as plain text or vice versa. Second, in the context of semi-structured content, a payload may contain encrypted and unencrypted content. In both cases, the payload potentially offers interesting insights that can be inspected. In the vast amount of traffic at hand, however, finding these specific sessions or packets is nearly impossible by the human eye. An automatic approach to the aforementioned problem, therefore, can solve the problem of finding the needle in the haystack while at the same time complementing Moloch’s missing abilities to integrate packet inspection directly.

By applying statistical measures (*cf.* Section 2.5) that can distinguish different degrees of randomness in a sequence, the degree of encryption can be determined. Using these measures as an input vector for the training of any kind of clustering algorithm can then reveal payloads that contain potentially interesting content.

This concept was planned as an additional feature in the application but was never implemented fully and is therefore not part of the final product due to the following reasons: First, the selection of suitable training data and the generation of labeled data was cumbersome and difficult and would have required its own selection procedures and algorithms. Second, developing, training and finalizing a model good enough to produce helpful results would have required a lot of additional time and research into this specific topic. Third, the integration with Moloch would have been difficult and potentially problematic regarding the speed of the application as a whole, as the processing speed for both the reading of PCAP files and the tagging of large amounts of data is very slow.

Figure 4.5: Sequence diagram of the *Kronos* user agent data retrieval process

Figure 4.6: Sequence diagram of the *Kronos* link generation processFigure 4.7: The setup of the *Kronos* Docker containers

Chapter 5

Evaluation

This chapter describes how *Kronos* provides functionality needed to reach the project goals and use cases (*cf.* Subsection 3.3.1) and how it meets the described functional (Subsection 3.3.3) and quality requirements (Subsection 3.3.4).

For the evaluation of *Kronos*' data analysis features, different data sets were generated. Each scenario described in Section 5.1 partially or fully mimics a use case or a requirement described in Section 3.3. To evaluate user-friendliness, usability and practicability of *Kronos*, a survey was conducted on the basis on the System Usability Scale (SUS) [51] (*cf.* Appendix C).

5.1 Evaluation Data Set

To evaluate whether *Kronos* adheres to the specified use cases and requirements, several scenarios were simulated in a controlled environment. By using a VPN-based setup in a local network (*cf.* Figure 5.1), all traffic passing through a local router was logged and stored. Based on this setup, network traffic was created for the following scenarios using three different physical devices: iPhone 6s (A), iPad 10 (B) and MacBook Pro 2019 (C).

Table 5.1: Different network traffic scenarios

Name	Description	Involved Software	Devices
Scenario 1a	One device is actively using multiple messengers	Messengers (WhatsApp, Telegram, iMessage, Facebook)	A
Scenario 1b	Multiple devices are actively using multiple messengers in parallel	Messengers (WhatsApp, Telegram, iMessage, Facebook)	A, B, C
Scenario 2	Multiple devices actively use all kinds of software	Messengers, Applications, system related activities	A, B, C

Continued on next page

Table 5.1 – *Continued from previous page*

Name	Description	Involved Software	Devices
Scenario 3	A device loses its WLAN connection and later regains it	Normal device use	A
Scenario 4	The iOS devices in the network are updating their operating system	Apple Update	A, B, C
Scenario 5a	A device is sending large amounts of data via various messengers	Messengers(WhatsApp, Facebook)	A
Scenario 5b	Multiple devices are sending large amounts of data via various messengers	Messengers(WhatsApp, Facebook)	A, B, C

The evaluation of *Kronos*' features and its usability is conducted in the following sections using the evaluation data set and following the different use-cases and requirements defined in Chapter 3.

5.2 Use Cases

In this section, *Kronos*' applicability in the various use cases as described in Subsection 3.3.1 is evaluated and documented.

5.2.1 Insights into applications or devices in use

Devices in use

Kronos can provide deep insights into the device classes and operating systems that are present within the data: It offers an array of features that address this issue by looking at the user-agents in the data.

When in the main view, the panel named *OS* (*cf.* Figure 5.2) on the right-hand side, displays user-agent related information. It offers three main features, accessible by clicking on the different list items: First, the top item, *User Agents*, gives an overview of the user-agents present in the data. It splits the user-agent information into four categories: "Device Class", "Device Name", "Operating System Class" and "Operating System Name" (*cf.* Figures 5.3, 5.4, 5.5, 5.6). By clicking the respective tabs, the absolute number of appearances is displayed. By using the provided feature described above, general information about the presence or absence of certain types of devices can be determined. The second item, *Apple Server Bags*, offers insights specifically on Apple's "Server Bag" user-agents by displaying the appearance of these user-agents over time. This feature

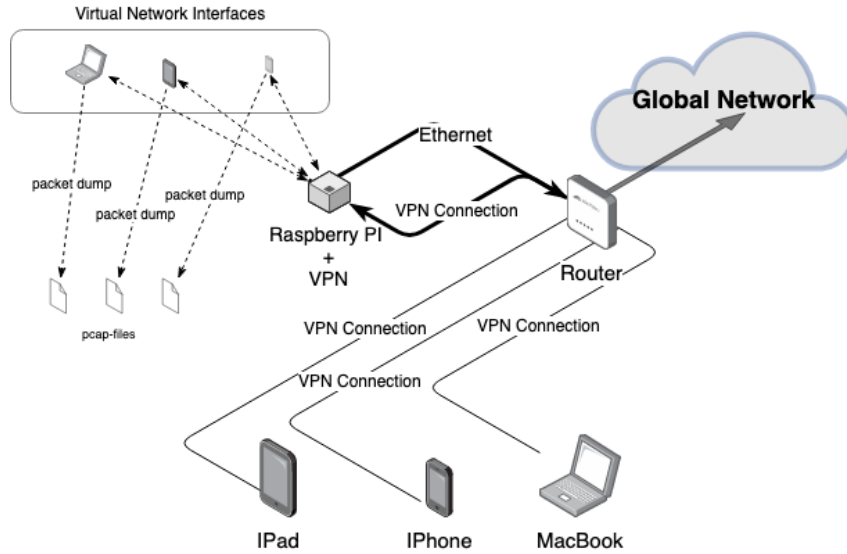


Figure 5.1: By using a VPN in a local network, the traffic of each connected device, can be identified and observed. All traffic can be recorded at the VPN server.

allows (i) to get an overview of the different Apple devices present over time and (ii) to deduce the actual amount of Apple devices present by taking the OS update patterns into account (*cf.* Figure 5.7). Third, all other items following the first two are specific filters configured by the user offering insights into different user-agents present. As can be seen in Figure 5.2, several items are listed below *Apple Server Bags*, which indicate additional user-defined user-agent filters. Clicking on any of these items displays a time-axis based view that gives an overview of the user-agents matching the specified regular expression over time as can be seen in Figure 5.9. The regular expressions configured can be viewed by clicking the *Show Config*-Button on the top right (*cf.* Figure 5.8).

Looking at the evaluation data from all scenarios, the tab *Device Class* shows the presence of at least two different classes: A *Mobile/Phone* and a *Desktop* class. The class *Unknown* accounts for a another device class, that could not be parsed properly. The devices A, B and C used in the scenarios are therefore recognizable (*cf.* Figure 5.3). This finding is supported by the other tabs as well, as shown in Figures 5.4, 5.5 and 5.6.

The OS panel also reveals the presence of three different types of server bag user-agents, hence, of three different Apple operating systems, as shown in Figure 5.7. For example, if the release date of the Apple iOS update 13.2.3 (18.11.2019 [52]) is taken into consideration, it can be concluded that (at least) one Mac computer and one mobile device were present. Hence, two of the three lanes belong to the same physical device. Therefore, two of the three devices could be distinguished from each other. The presence of the third Apple device is not revealed by *Kronos*.

Of all the regular expressions defined in the *Kronos* configuration file (*cf.* Figure 5.8), only the filters for *Apple* and *Mozilla* contain data when applied to the evaluation data. This reveals the absence of any Linux or Android operating system with a high probability. Based on the data filtered by *Kronos*, it can be seen that (i) the browser Chrome was used by a Mac computer with the operating system 10.15.1 during the entire time (Figure 5.10) and (ii) a mobile Apple device performed an update (Figure 5.11).

Kronos - Network Data Inspection Tool

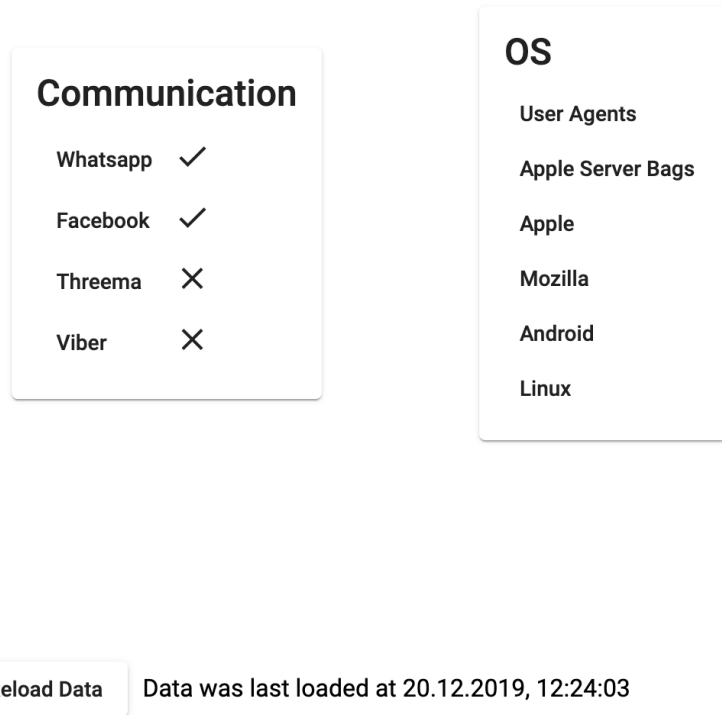


Figure 5.2: The main view of the application: The feature panel on the left, the feature panel *OS* on the right

Device Class	Device Name	Operating System Class	Operating System Name
Device Class			Count
Unknown			4
Mobile			3
Phone			2
Desktop			1

Figure 5.3: Overview of the user agents present in the evaluation data set. Tabs indicate different categories: “Device Class”, “Device Name”, “Operating System Class” and “Operating System Name”

Applications in use

Kronos also offers the possibility to inspect traffic related to specific, freely configurable DNS hostnames. This feature gives insights into the applications that generated the

Device Class	Device Name	Operating System Class	Operating System Name
Device Name		Count	
Unknown		4	
Apple iOS Device		3	
Apple iPhone		2	
Apple Macintosh		1	

Figure 5.4: The user-agent overview category “Device Names” offering insights into the different device names present in the data

Device Class	Device Name	Operating System Class	Operating System Name
OS Class		Count	
Unknown		6	
Mobile		3	
Desktop		1	

Figure 5.5: The user-agent overview category “Operating System Class” offering insights into the different operating system classes present in the data

Device Class	Device Name	Operating System Class	Operating System Name
OS Name		Count	
Unknown		6	
Darwin (iOS)		3	
Mac OS X		1	

Figure 5.6: The user-agent overview category “Operating System Name” offering insights into the different operating system names present in the data

traffic observed, since DNS requests are unencrypted and the hostnames are revealing in plain-text (*e.g.* `chat.cdn.whatsapp.net`).

By accessing the left-hand panel, *Communication*, from the main view (*cf.* Figure 5.2), the user is able to click on any of the items in the list, each connected to a specific regular expression defined in the *Kronos* configuration file (*cf.* Figure 5.8). A cross icon indicates right away the absence of data matching the pattern, while a tick icon indicates that data is available. By clicking on a specific item on this list, *Kronos* offers two views, each showing detailed information on either the DNS request frequency over time (*cf.* Figures 5.12 and 5.14) or the the data flow from or to the matching destinations (*cf.* Figures 5.13 and 5.15).

From *Kronos*’ main view, it can be deduced, that most likely neither Threema nor Viber (chatting apps) were used in the data. However, it can be seen that the data contains traffic from and to WhatsApp as well as Facebook. Further, by inspecting both applications, it is observed that DNS hostnames names such as `chat.cdn.whatsapp.net` or `edge-chat.facebook.com` were used. Most likely, both applications were, therefore, used for their messenger services. This discovery matches the messengers used to generate

iOS Server Bag Appearance

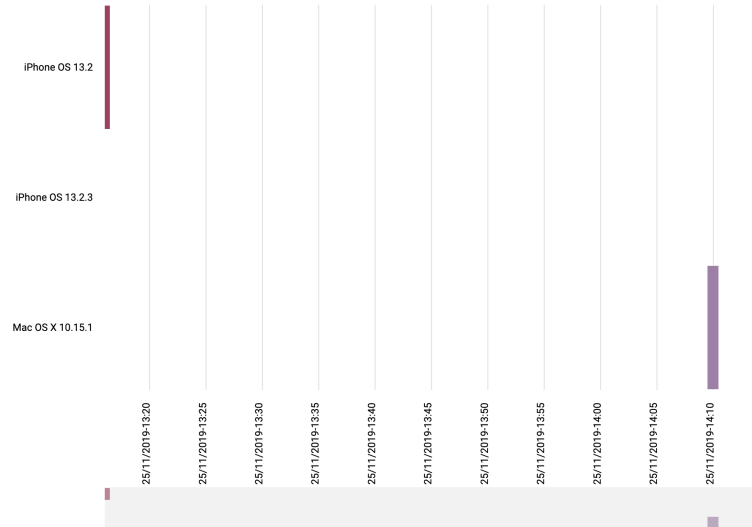


Figure 5.7: The present Apple devices are displayed over time. By taking the fact into account that iOS 13.2.3 was available, it can be concluded that the two devices displayed as iOS 13.2 and iOS 13.2.3 are one and the same device

the data: In scenarios 1a and 1b, Facebook, as well as WhatsApp, were both used as messengers. It is likely, that if present in the *Kronos* configuration file, the presence of Telegram could have been detected as well. As an additional finding, the hostname “instagram.c10r.facebook.com” visible in Figure 5.12, reveals not only the presence of Instagram within the data, but also the fact that Instagram most likely uses servers with *facebook.com* domains ¹.

In summary, *Kronos* offers an inside look into the presence of devices, their Operating Systems (OS) as well as the use of specific applications. If the insights resulting from all different *Kronos* features described above are combined, the devices, their OSes, specific activities and applications from the scenarios 1a/b (different messengers), scenario 2 (detection of devices in parallel) and scenario 4 (iOS updates), can be deduced without knowing the data beforehand. The extraction of device-specific information done by *Kronos*, therefore, matches the use cases and fulfills the project goals.

5.2.2 Automated identification of unencrypted network traffic

As mentioned in Section 4.6, the automatic payload-based classification of network traffic into encrypted and unencrypted traffic, was not implemented due to various reasons stated in Section 4.6 and 6. An evaluation of *Kronos*’ ability to determine encrypted traffic from unencrypted traffic is therefore not possible in the application’s current state.

¹Instagram was sold to Facebook in 2012. Initially, it was decided that Instagram should be run as an independent service [53]. The data, however, shows otherwise.

Current Kronos Config

```

{
  "kronosQueryConfiguration": {
    "networkTrafficConfigurationList": [
      {
        "description": "Whatsapp",
        "regex": ".*whatsapp.*"
      },
      {
        "description": "Facebook",
        "regex": ".*facebook.*"
      },
      {
        "description": "Threema",
        "regex": ".*threema.*"
      },
      {
        "description": "Viber",
        "regex": ".*viber.*"
      }
    ],
    "userAgentConfigurationList": [
      {
        "description": "Apple",
        "regex": ".*[aA]pple.*"
      },
      {
        "description": "Mozilla",
        "regex": ".*[mM]ozilla.*"
      },
      {
        "description": "Android",
        "regex": ".*[aA]ndroid.*"
      },
      {
        "description": "Linux",
        "regex": ".*[lL]inux.*"
      }
    ]
  }
}

```

Figure 5.8: The contents of the *Kronos* configuration file written in YAML: Listing all user-agent and communication-related regular expressions.

5.2.3 Identify patterns or anomalies

From the views provided by *Kronos* through the time-dependent data- and DNS-based visualizations, more information can be extracted: By focusing on the temporal dimensions of the network traffic patterns through the features provided by the *DNS Request* and the *Data Flow to Servers* tabs, activity patterns can be identified. At the same time, unexpected activities that deviate from the normal pattern give additional insights.

Taking Facebook as an example (*cf.* Figures 5.12 and 5.13), it can be observed: (i) during which times the most traffic was transmitted (start and end dates as well as periods of high and low activity), (ii) during which periods no traffic took place, (iii) which specific servers were requested with what frequency and (iv) which periods had the highest data transmissions, indicating *e.g.* media transfers.

Looking at the *Data Flow to Servers* tab (*cf.* Figure 5.13), it can be seen that the most data was transmitted to/from the servers *edge-chat.facebook.com* and *upload.facebook.com*. By the names of the servers it can be concluded with high probability, that Facebook's

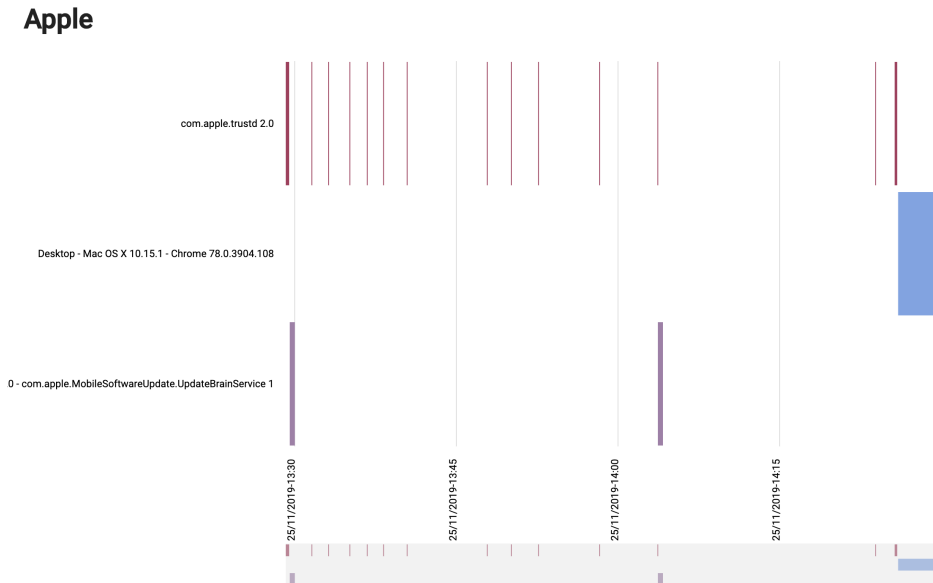


Figure 5.9: The timeline for the user-agents matching the specified regular expression. What user-agents *Kronos* was filtering for is defined by the *Kronos* configuration file (cf. Figure 5.8)

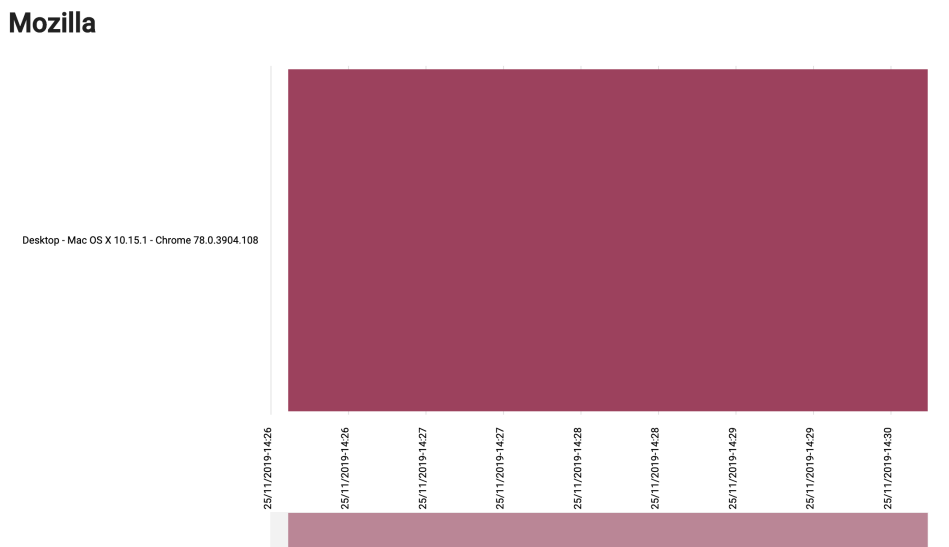


Figure 5.10: *Kronos* filtering the data for any user-agents matching the regular expression *.*[mM]ozilla.** (cf. Figure 5.8)

chat functionality was used and that, during certain periods, data uploads took place. This deduction is supported further by the fact that the upload and the chat servers have spikes in the data transmission at the same time. Similar conclusions can be drawn for WhatsApp (cf. e.g. Figure 5.14).

As observed in Figures 5.12, 5.13, 5.14 and 5.15, *Kronos* allows deep insights into the details of the network traffic itself and helps to identify usage patterns such as absence, presence and times of higher and of lower activity. The observations made through *Kro-*

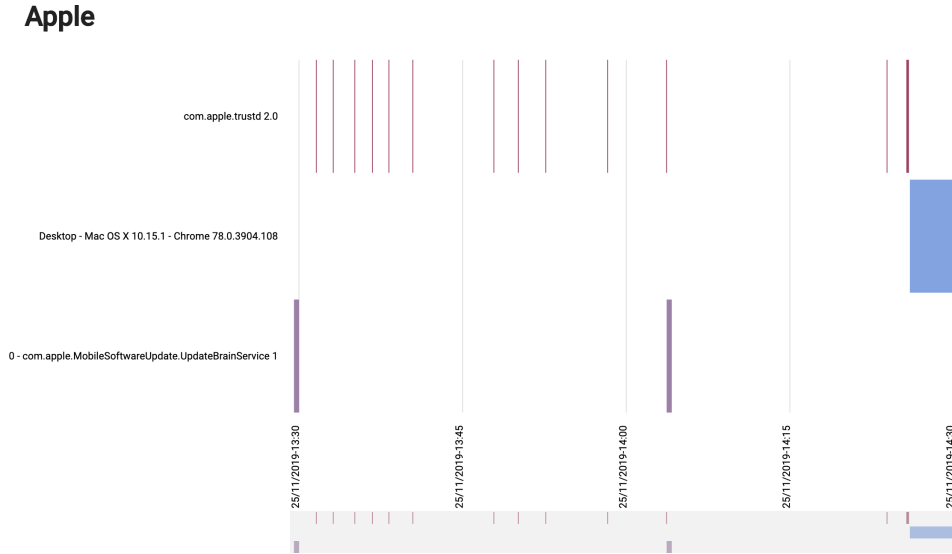


Figure 5.11: *Kronos* filtering the data for any user-agents matching the regular expression `.*[aA]pple.*` (cf. figure 5.8)

nos, allow the reconstruction of the evaluation data setup of scenarios 1a and 1b (used messengers), scenario 2 (multiple devices in parallel), scenario 3 (loss of WLAN connection) and scenarios 5a and 5b (data transmission) with a high level of accuracy. They show that *Kronos* can detect activities and patterns regarding different ways of communication on different levels of detail.

5.3 Evaluation of Requirements

As described in Section 3.3, the use cases lead to several requirements (cf. Subsections 3.3.3 and 3.3.4), that are evaluated in this Section.

5.3.1 Functional Requirements

Regarding the functional requirements specified in Subsection 3.3.3, *Kronos* can meet the expected features in the following way:

1. **Visualization of DNS requests for hostname over time:** By clicking on any item in the *Communication* panel, all DNS requests sent to a DNS server for a certain hostname matching the specific regular expression defined beforehand in the *Kronos* configuration file are displayed over time (cf. Figures 5.12 and 5.14) in the *DNS Requests* tab. By using the control at the bottom, specific windows in time can be inspected in more detail.
2. **Visualization of the amount of transmitted data for hostname over time:** The same DNS hostnames mentioned before, are visualized in the *Data Flow to*

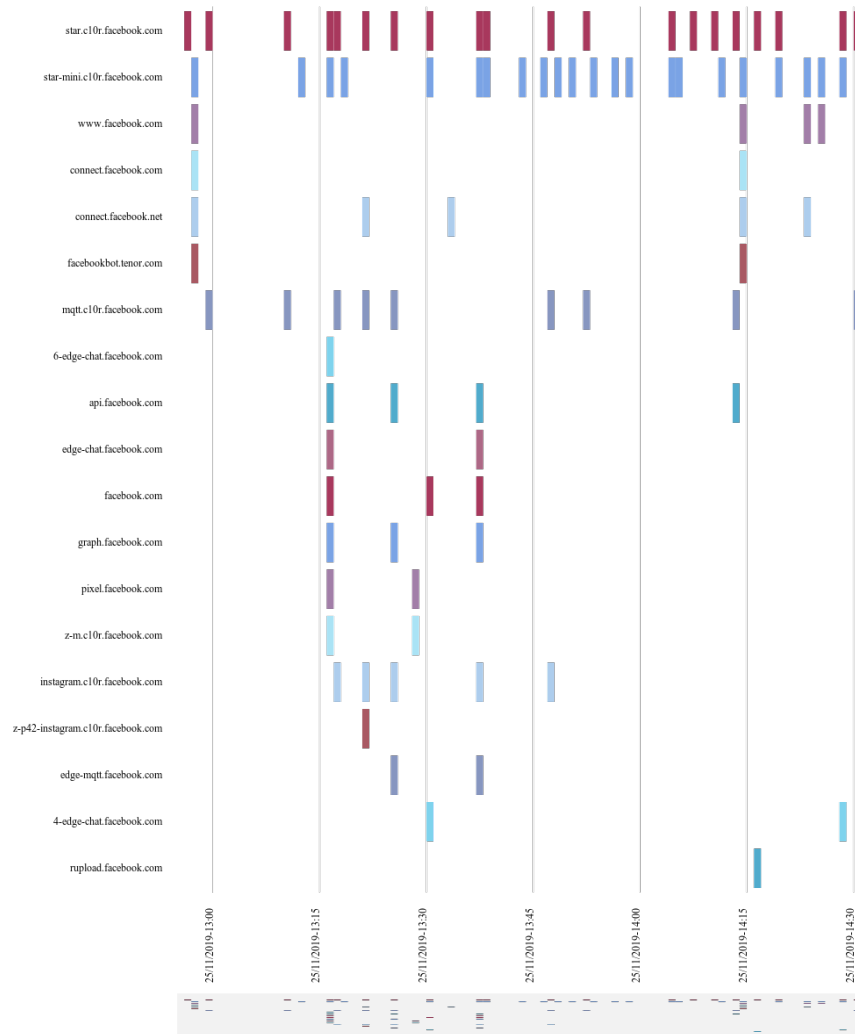


Figure 5.12: Calls to different DNS host names (Y-axis) that match the user specified regular expression for Facebook over time (X-axis)

Servers tab regarding the data transmitted from or to any IP returned by the DNS server within a specific time interval (*cf.* Figures 5.13 and 5.15). Hovering over a specific bar in the visualization reveals the exact amount of bytes that were transmitted. The control at the bottom facilitates the same manipulation of the visualization as mentioned before.

3. **Display of aggregated user agent information:** By clicking on the first item in the *OS* panel, the frequencies of different user-agent related information is displayed: The four tabs “Device Class”, “Device Name”, “Operating System Class” and “Operating System Name” reveal the specific devices present in the data (Figure 5.3).
4. **Visualization of user agent appearance over time:** Also in the *OS* panel, the second and all the following items allow to inspect specific user-agents over time: Similar to the visualizations from DNS requests and data flow, the appearance of user-agents matching the regular expressions defined in the *Kronos* configuration file are displayed over time (*cf.* Figures 5.10 and 5.11). A control at the bottom



Figure 5.13: Data flow to IPs received by different DNS host names (Y-axis) that match the user specified regular expression for Facebook over time (X-axis)

allows inspecting specific time windows of the data.

5. **Automated detection of (un-)encrypted traffic:** This functionality is not represented by any feature in *Kronos*. Due to the reasons stated in section 4.6, the development of encryption detection was stopped and is not part of the final application.

5.3.2 Quality Requirements

Regarding the quality requirements specified in subsection 3.3.4 *Kronos* addresses the different aspects as follows:

1. **Usability:** Figure 5.2 shows that *Kronos* offers first pieces of information about the absence of certain messengers at first sight. All other functionality is only one

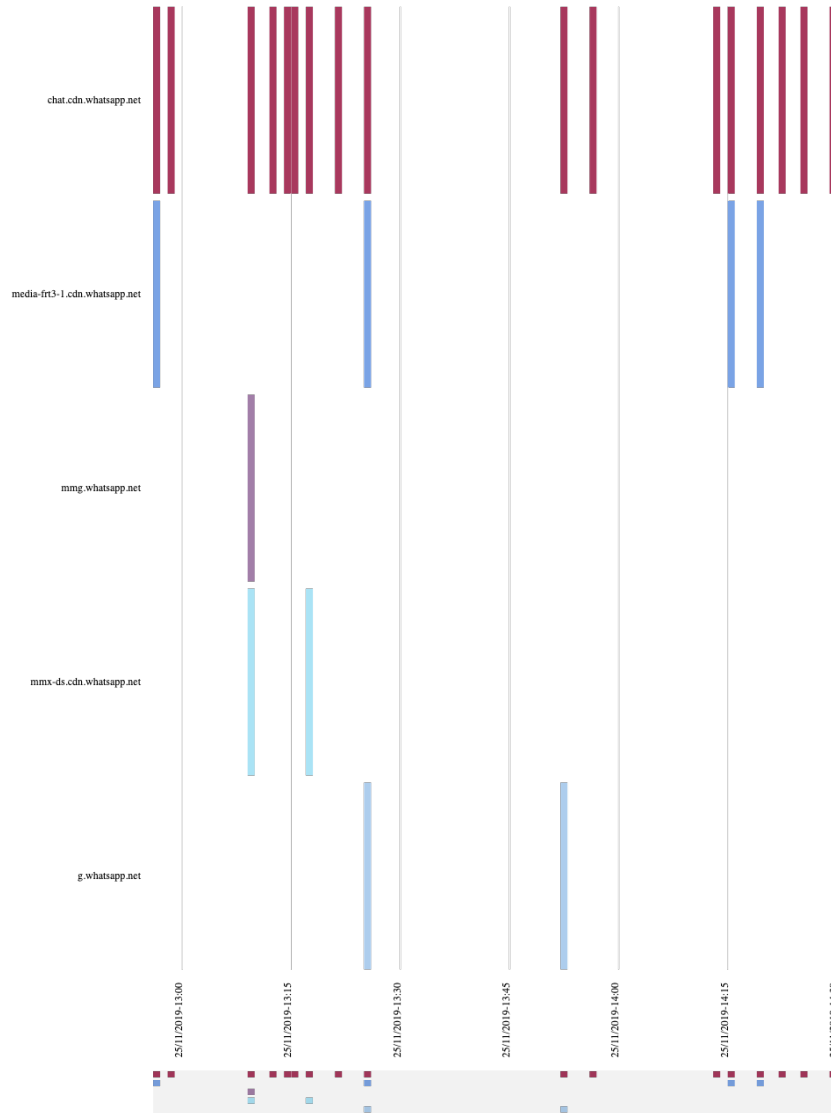


Figure 5.14: Calls to different DNS servers (Y-axis) that match the user specified regular expression for WhatsApp over time (X-axis)

click away, offering easy to interpret visualizations that help to understand the network traffic, while at the same time allowing a maximum amount of configurability regarding all main features. According to the conducted SUS-based evaluation², *Kronos* obtains an average SUS of 93.33%. As can be seen in Figure 5.16, the obtained SUS score is interpreted as excellent to “best imaginable”.

2. **Maintainability:** *Kronos* is platform-independent due to its deployment with Docker. Its source code is well documented and adheres to established software engineering principles, such as cohesion, loose coupling and decomposition. It should, therefore, be easily maintainable and deployable.

3. **Future developments:** By decoupling the front-end and the back-end and by

²Size of population: Unknown, Sample size: 3

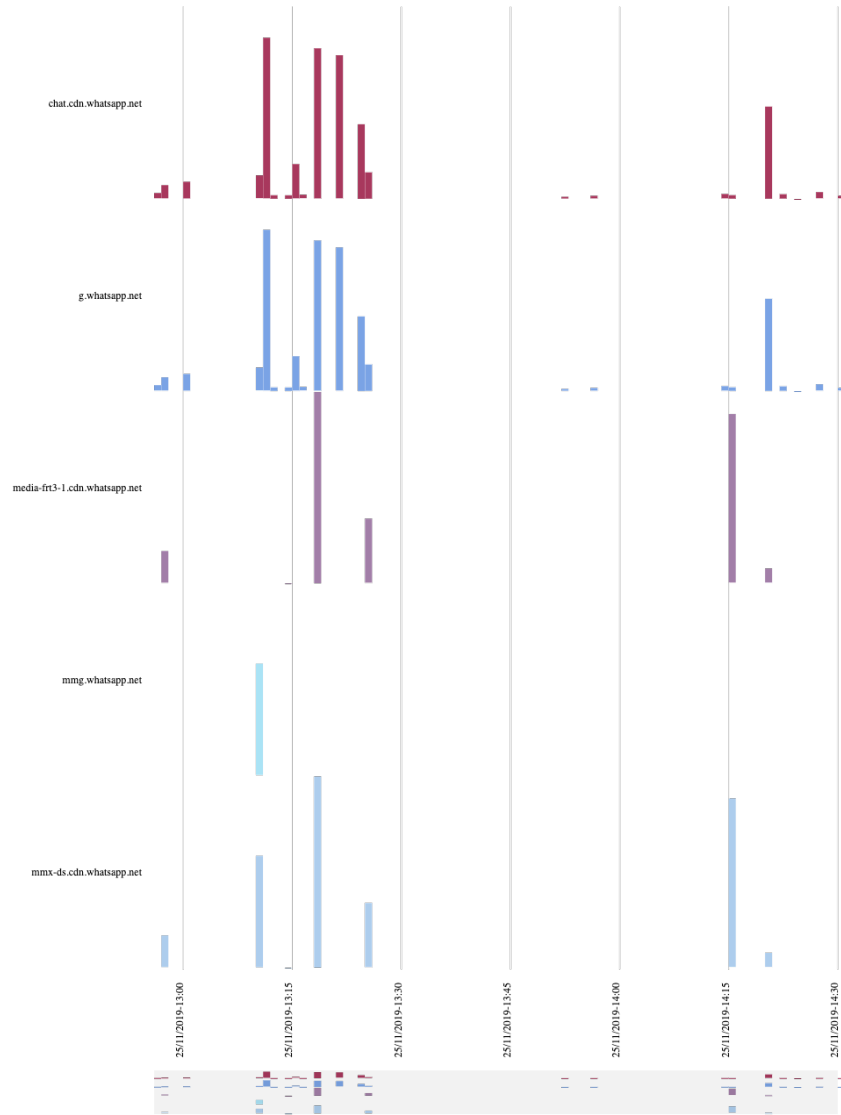


Figure 5.15: Data flow to IPs received by different DNS servers (Y-axis) that match the user specified regular expression for WhatsApp over time (X-axis)

decomposing *Kronos*' code into a hexagonal architecture, new features can be added easily and with minimal adaptations of the code. Basic changes can even be made without having to change the actual source code by adapting a configuration file. Therefore, *Kronos* is easily improvable and adaptable to new requirements.

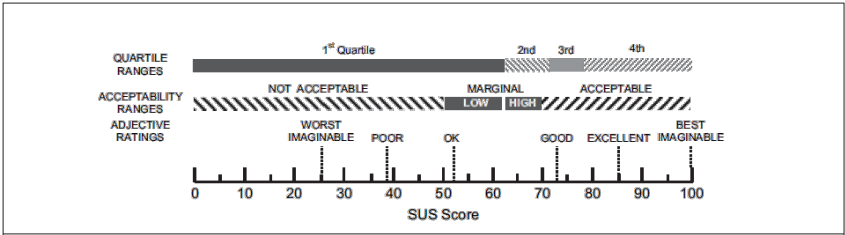


Figure 5.16: Chart for the interpretation of the non-linear system usability scale [54]

Chapter 6

Summary and Future Work

This report describes a forensic network data inspection tool developed for six months for and in collaboration with KAPO ZH. The KAPO ZH specific use cases were assessed and both functional and quality requirements were derived from them. *Kronos* was designed aiming to profit as much as possible from the existing infrastructure at KAPO ZH while providing the flexibility needed to add value to systems currently in use. The resulting application implements modern application frameworks, programming paradigms and languages to offer high maintainability in the future, which was also a requirement from KAPO ZH. The integration of a configuration file further improves *Kronos*' adaptability, allowing future code maintainers to make basic changes and amendments without having to deal with the actual source code in the first place. A modern and intuitive UI was designed that aims to create a low entry barrier; even for non-expert users. The UI includes highly customized visualizations that provide insight into the data that was not possible with the applications currently in use at KAPO ZH.

The evaluation of the application shows that most of the project goals could be met. The use cases described in the project planning phase were implemented in *Kronos*, as shown based on a customized synthetic test data set. The automated detection of encrypted or unencrypted traffic forms a slight exception. While the initial groundwork has been done from a research perspective and a proposal for an approach has been formulated, the feature could not be integrated into the application due to a lack of labeled training data. The qualitative goals of usability and maintainability were also met, as was shown using the SUS and a description of the systems development guidelines and architectural considerations.

6.1 Future Work

Kronos features a configuration file that simplifies changes or additions to the network traffic that should be inspected. This already provides a high amount of flexibility for the application and enables fast and easy adaptations. Future versions of the application could extend this concept further to make the application fully configurable. The configuration file could then include, for example, the Moloch fields to search in Elastic with

the corresponding regular expressions. Since *Kronos* is used by multiple users, the implementation of a per-user configuration could also be beneficial. This would allow each user to maintain his own individual *Kronos* setup that is tuned to his or her individual use cases. A possibility for sharing configurations between users could also be advantageous.

Initial research into the automated detection of (un-)encrypted traffic was done within the scope of this project. However, the feature could not be integrated into the application as there was not sufficient labeled test data to tune the parameters of an encryption detection approach. Thus, such an encryption detection feature could be implemented in future versions of *Kronos*, which can further include the automated tagging of the corresponding sessions in Moloch.

The System Usability Score (SUS) assessed in the evaluation phase of this project shows that the application developed is already usable and intuitive. However, further improvements are always possible to fine-tune the UI to its users. With sufficient expenditure of time, the custom charts developed for this project could be optimized further, with regards to speed and the degree to which they adapt to the data. In addition to this, further custom charts could be implemented that provide even more insight into the data.

Finally, future versions of *Kronos* could implement further sophisticated analysis of network traffic data. This may require an in-depth analysis of the individual network packets and may thus diverge from the current approach of using Moloch's packet pre-processor and Elasticsearch to maintain scalability. It would, however, allow the extraction of further interesting data, such as e-mail addresses used or the search for specific keywords in unencrypted payloads.

Bibliography

- [1] Wikipedia, “Digital forensics,” 2019. [Online]. Available: https://en.wikipedia.org/wiki/Digital_forensics (Accessed 2019-12-13).
- [2] G. G. Mark Reith, Clint Carr, “An Examination of Digital Forensic Models,” *International Journal of Digital Evidence*, vol. 1, no. 3, 2002.
- [3] B. Carrier, “Defining Digital Forensic Examination and Analysis Tools,” in *Digital Forensics Research Workshop 2002*, NY, 2002.
- [4] G. Palmer, “A Road Map for Digital Forensic Research,” in *The Digital Forensic Research Conference*, vol. 24, no. 3, Utica, NY, 2001, pp. 709–719.
- [5] INCIBE-CERT, “Understanding industrial network traffic, dissectors and Lua and Kaitai.” [Online]. Available: <https://www.incibe-cert.es/en/blog/understanding-industrial-network-traffic-dissectors-and-lua-and-kaitai> (Accessed 2019-07-17).
- [6] L. Baumann, “An Explorative Research into Encrypted Data Analysis,” Independent Study, University of Zürich, 2019.
- [7] Alessandro Balzano, “How to write generic dissectors in Wireshark | Diplomats can code too!” 2019. [Online]. Available: <http://wintermade.it/blog/posts/how-to-write-generic-dissectors-in-wireshark.html> (Accessed 2019-07-17).
- [8] K. Gold, “Active vs passive network monitoring,” 2019. [Online]. Available: <https://www.exfo.com/en/resources/blog/active-passive-network-monitoring/> (Accessed 2019-12-13).
- [9] Wikipedia, “Hypertext Transfer Protocol Secure,” 2019. [Online]. Available: https://de.wikipedia.org/wiki/Hypertext_Transfer_Protocol_Secure (Accessed 2020-01-07).
- [10] Wikipedia, “TOR (anonymity network),” 2020. [Online]. Available: [https://en.wikipedia.org/wiki/Tor_\(anonymity_network\)](https://en.wikipedia.org/wiki/Tor_(anonymity_network)) (Accessed 2020-01-07).
- [11] K. Finley, “Half the Web Is Now Encrypted. That Makes Everyone Safer,” 2017. [Online]. Available: <https://www.wired.com/2017/01/half-web-now-encrypted-makes-everyone-safer/#> (Accessed 2019-12-20).

- [12] L. Foundation, “Let’s Encrypt Stats - Let’s Encrypt - Free SSL/TLS Certificates,” 2019. [Online]. Available: <https://letsencrypt.org/stats/{#}growth> (Accessed 2019-12-20).
- [13] A. Topuzoğlu, *Topics in Geometry, Coding Theory and Cryptography*, 1st ed., A. Garcia and H. Stichtenoth, Eds. Springer Netherlands, 2007.
- [14] Wikipedia, “Entropy (Information theory),” 2020. [Online]. Available: [https://en.wikipedia.org/wiki/Entropy{ }\(information{ }theory\)](https://en.wikipedia.org/wiki/Entropy{ }(information{ }theory)) (Accessed 2019-12-12).
- [15] R. W. Hamming, “Coding and Information Theory,” *Englewood Cliffs NJ: Prentice-Hall*, pp. 104–108, 1980.
- [16] C. E. Shannon and W. Weaver, *The Mathematical Theory of Communication*. Urbana, IL: The University of Illinois Press, 1949.
- [17] J. Walker, “Ent: A Pseudorandom Number Sequence Test Program,” 2008. [Online]. Available: <https://www.fourmilab.ch/random/> (Accessed 2019-12-12).
- [18] Wikipedia, “Chi-squared Test,” 2020. [Online]. Available: <https://en.wikipedia.org/wiki/Chi-squared{ }test> (Accessed 2019-12-12).
- [19] D. E. Knuth, *The Art of Computer Programming*, 3rd ed. Reading, Massachusetts: Addison-Wesley Longman, 1997, vol. Volume 2.
- [20] J. H. Mathews, “Monte Carlo Pi,” 2005. [Online]. Available: <http://mathfaculty.fullerton.edu/mathews/n2003/montecarlopi.html> (Accessed 2019-12-12).
- [21] Wikipedia, “OS-level virtualization,” 2019. [Online]. Available: <https://en.wikipedia.org/wiki/OS-level{ }virtualization> (Accessed 2019-12-13).
- [22] CloudOye, “Operating System Level Virtualization.” [Online]. Available: <https://www.cloudoye.com/wiki/o/operating-system-level-virtualization> (Accessed 2019-12-12).
- [23] P. Christensson, “Web Application Definition,” 2014. [Online]. Available: <https://techterms.com/definition/web{ }application> (Accessed 2019-12-13).
- [24] Tranalyzer, “Tranalyzer.” [Online]. Available: <https://tranalyzer.com/about{#}theanteater> (Accessed 2019-12-18).
- [25] CyberChef, “CyberChef: The Cyber Swiss Army Knife - a web app for encryption, encoding, compression and data analysis.” [Online]. Available: <https://github.com/gchq/CyberChef> (Accessed 2019-12-12).
- [26] Wireshark Foundation, “Wireshark.” [Online]. Available: <https://www.wireshark.org/> (Accessed 2019-12-12).
- [27] Elasticsearch, “What is Elasticsearch,” 2019. [Online]. Available: <https://www.elastic.co/what-is/elasticsearch> (Accessed 2019-12-18).
- [28] AOL, “Moloch.” [Online]. Available: <https://github.com/aol/moloch> (Accessed 2019-12-10).

- [29] Y. Mulonda and Noteworthy - The Journal Blog, "What is Docker? "In Simple English"," 2018. [Online]. Available: <https://blog.usejournal.com/what-is-docker-in-simple-english-a24e8136b90b> (Accessed 2019-12-13).
- [30] Docker, "What is a Container?" [Online]. Available: <https://www.docker.com/resources/what-container> (Accessed 2019-12-12).
- [31] Wikipedia, "Moloch." [Online]. Available: <https://en.wikipedia.org/wiki/Moloch> (Accessed 2020-1-2).
- [32] S. Eidgenossenschaft, "Dienst Überwachung Post- und Fernmeldeverkehr," 2019. [Online]. Available: <https://www.li.admin.ch/de/stats> (Accessed 2019-12-13).
- [33] TypeScript, "TypeScript - JavaScript that scales." [Online]. Available: <https://www.typescriptlang.org/index.html> (Accessed 2019-12-10).
- [34] Angular, "Angular - one framework. mobile & desktop." [Online]. Available: <https://angular.io> (Accessed 2019-12-10).
- [35] Stackoverflow, "Developer survey results 2018." [Online]. Available: <https://insights.stackoverflow.com/survey/2018#technology> (Accessed 2019-15-10).
- [36] Angular, "Angular material - material design components for angular." [Online]. Available: <https://material.angular.io> (Accessed 2019-12-10).
- [37] swimlane, "ngx-charts - declarative charting framework for angular!" [Online]. Available: <https://github.com/swimlane/ngx-charts> (Accessed 2019-12-10).
- [38] tutorialspoint, "Mvc framework - introduction." [Online]. Available: https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm (Accessed 2019-15-10).
- [39] Kotlin, "Kotlin." [Online]. Available: <https://kotlinlang.org> (Accessed 2019-15-10).
- [40] TechCrunch, "Kotlin is now google's preferred language for android app development." [Online]. Available: <http://techcrunch.com/2019/05/07/kotlin-is-now-googles-preferred-language-for-android-app-development/> (Accessed 2019-15-10).
- [41] Kotlin, "Using kotlin for server-side development." [Online]. Available: <https://kotlinlang.org/docs/reference/server-overview.html> (Accessed 2019-15-10).
- [42] Kotlin, "Hexagonal architecture." [Online]. Available: <https://kotlinlang.org/docs/reference/coroutines-overview.html> (Accessed 2019-15-10).
- [43] Ktor, "Ktor - easy to use, fun and asynchronous." [Online]. Available: <https://ktor.io> (Accessed 2019-15-10).
- [44] Koin, "Koin - what is koin?" [Online]. Available: <https://insert-koin.io> (Accessed 2019-15-10).

- [45] Elastic, “Java high level rest client.” [Online]. Available: <https://www.elastic.co/guide/en/elasticsearch/client/java-rest/master/java-rest-high.html> (Accessed 2019-15-10).
- [46] A. Cockburn, “Coroutines for asynchronous programming and more.” [Online]. Available: <https://web.archive.org/web/20180822100852/alistair.cockburn.us/Hexagonal+architecture> (Accessed 2018-8-22).
- [47] D. R. Prasanna, “Dependency injection,” *KU Digital Collections*, 2009.
- [48] C. H. . L. Baumann, “Github pr: Fix for ktor-client digest authentication.” [Online]. Available: <https://github.com/ktorio/ktor/pull/1386> (Accessed 2019-17-10).
- [49] Docker, “Overview of docker compose.” [Online]. Available: <https://docs.docker.com/compose/> (Accessed 2019-17-10).
- [50] M. Fowler, J. Highsmith *et al.*, “The agile manifesto,” *Software Development*, vol. 9, no. 8, pp. 28–35, 2001.
- [51] J. Brooke, “System Usability Scale (SUS).” [Online]. Available: <https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html> (Accessed 2019-12-20).
- [52] Wikipedia, “iOS version history,” 2019. [Online]. Available: <https://en.wikipedia.org/wiki/iOS{-}version{-}history{#}iOS{-}13> (Accessed 2019-12-20).
- [53] E. M. Rusli, “Facebook Buys Instagram for \$1 Billion,” 2012. [Online]. Available: <https://dealbook.nytimes.com/2012/04/09/facebook-buys-instagram-for-1-billion/> (Accessed 2019-12-30).
- [54] M. Rauer, “Quantitative Usability-Analysen mit der System Usability Scale (SUS),” 2011. [Online]. Available: <https://blog.seibert-media.net/blog/2011/04/11/usability-analysen-system-usability-scale-sus/> (Accessed 2019-12-21).
- [55] T. Nathan, “How To Use The System Usability Scale (SUS) To Evaluate The Usability Of Your Website - Usability Geek,” 2015. [Online]. Available: <https://usabilitygeek.com/how-to-use-the-system-usability-scale-sus-to-evaluate-the-usability-of-your-website/> (Accessed 2020-01-11).

Abbreviations

API	Application Programming Interface
CD	Compact Disc
MAP	Master Project
CLI	Command Line Interface
CORS	Cross Origin Ressource Sharing
CSS	Cascade Style Sheet
D3	Data-Driven Documents
DI	Dependency Injection
DNS	Domain Name Server
DoH	DNS over HTTPS
HTML	Hypertext Markup Language
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
IP	Internet Protocol
ISP	Internet Service Provider
JDK	Java Developement Kit
JRE	Java Runtime Enviroment
JS	JavaScript
JVM	Java Virtual Machine
KAPO ZH	Zürich Canontal Police or Kantonspolizei Zürich
NPM	Node Package Manager
OS	Operating System
PCAP	Packet Capture File Format
REST API	Representational State Transfer Application Programming Interface
SUS	System Usability Scale
SVG	Scalable Vector Graphics
TOR	The Onion Routing or The Onion Router
TS	TypeScript
UI	User Interface
VPN	Virtual Private Network
WLAN	Wireless Local Area Network
YAML	Yet Another Markup Language or YAML Ain't Markup Language

List of Figures

2.1	The two dimensions of network traffic monitoring: Active vs. Passive monitoring and Live vs. Ex Post monitoring [8].	4
2.2	The percent of pageloads over HTTPS in the years 2014 to 2019 by different user origins [12].	5
2.3	An abstract view of the functionality provided by the docker environment through the docker daemon and the different containers (here A - F) that hold different applications [30].	9
3.1	A comparison from 2018 between all Swiss cantons regarding their surveillance activities [32].	13
3.2	Reasons for surveillance for the canton of Zurich in 2018 [32].	13
3.3	Comparison of different kinds of surveillance jobs issued in Switzerland in the years 2014 to 2018 [32].	14
3.4	An abstract view of the <i>Kronos</i> system topology	16
3.5	An abstract view of a <i>Kronos</i> setup with multiple tenants and multiple Moloch-Elasticsearch ensembles. The data flow for each client is shown in a different color.	17
4.1	An overview of the building blocks of an Angular application [34]	21
4.2	An abstract view of the <i>Kronos</i> server application's hexagonal architecture. Data flow in request and response direction is shown in red and blue respectively.	24
4.3	Sequence diagram of the <i>Kronos</i> login process and session management . .	26
4.4	Sequence diagram of the <i>Kronos</i> communication data retrieval process . .	27
4.5	Sequence diagram of the <i>Kronos</i> user agent data retrieval process	31
4.6	Sequence diagram of the <i>Kronos</i> link generation process	32
4.7	The setup of the <i>Kronos</i> Docker containers	32

5.1	By using a VPN in a local network, the traffic of each connected device, can be identified and observed. All traffic can be recorded at the VPN server.	35
5.2	The main view of the application: The feature panel on the left, the feature panel <i>OS</i> on the right	36
5.3	Overview of the user agents present in the evaluation data set. Tabs indicate different categories: “Device Class”, “Device Name”, “Operating System Class” and “Operating System Name”	36
5.4	The user-agent overview category “Device Names” offering insights into the different device names present in the data	37
5.5	The user-agent overview category “Operating System Class” offering insights into the different operating system classes present in the data	37
5.6	The user-agent overview category “Operating System Name” offering insights into the different operating system names present in the data	37
5.7	The present Apple devices are displayed over time. By taking the fact into account that iOS 13.2.3 was available, it can be concluded that the two devices displayed as iOS 13.2 and iOS 13.2.3 are one and the same device .	38
5.8	The contents of the <i>Kronos</i> configuration file written in YAML: Listing all user-agent and communication-related regular expressions.	39
5.9	The timeline for the user-agents matching the specified regular expression. What user-agents <i>Kronos</i> was filtering for is defined by the <i>Kronos</i> configuration file (<i>cf.</i> Figure 5.8)	40
5.10	<i>Kronos</i> filtering the data for any user-agents matching the regular expression <i>.*[mM]ozilla.*</i> (<i>cf.</i> Figure 5.8)	40
5.11	<i>Kronos</i> filtering the data for any user-agents matching the regular expression <i>.*[aA]pple.*</i> (<i>cf.</i> figure 5.8)	41
5.12	Calls to different DNS host names (Y-axis) that match the user specified regular expression for Facebook over time (X-axis)	42
5.13	Data flow to IPs received by different DNS host names (Y-axis) that match the user specified regular expression for Facebook over time (X-axis)	43
5.14	Calls to different DNS servers (Y-axis) that match the user specified regular expression for WhatsApp over time (X-axis)	44
5.15	Data flow to IPs received by different DNS servers (Y-axis) that match the user specified regular expression for WhatsApp over time (X-axis)	45
5.16	Chart for the interpretation of the non-linear system usability scale [54] . .	46

List of Tables

5.1	Different network traffic scenarios	33
-----	---	----

Appendix A

Kronos Github Repository

The code, the documentation and all build instructions for *Kronos* can be found on Github via the following link: <https://github.com/cyrillhalter/kronos.git>. In order to get access please contact cyrill.halter@me.com. The repository is maintained by **cyrillhalter** and **HolzmanoLagrene**. For more information and access to the repository please see the *README*-file.

Appendix B

Contents of the CD

The CD attached to this report contains the following files (file names in brackets):

1. Report Master Project (MAP) (kronos-project-report.pdf)
2. Report Master Project (MAP) L^AT_EX source code (kronos-project-report.zip)
3. Reference to *Kronos* Github Repository (kronos-github-repository-reference.txt)
4. *Kronos* source code as per 13.01.2020 (kronos-github-repository.zip)
5. Intermediate Presentation (intermedpresentation.pptx)
6. Final Presentation (finalpresentation.pptx)

Appendix C

SUS Evaluation Form

This appendix includes the empty, as well as the three completed questionnaires. The SUS is calculated based on the ten questions of the first page (for instructions please consult *e.g.* <https://usabilitygeek.com/how-to-use-the-system-usability-scale-sus-to-evaluate-the-usability-of-your-website/> [55]). The second page contains additional questions including two open questions for direct feedback.

Evaluation Tool “Kronos”

	Stimme zu	Stimme nicht zu
1. Ich kann mir sehr gut vorstellen, das System regelmäßig zu nutzen.	12345	
2. Ich empfinde das System als unnötig komplex.	12345	
3. Ich empfinde das System als einfach zu nutzen.	12345	
4. Ich denke, dass ich technischen Support brauchen würde, um das System zu nutzen.	12345	
5. Ich finde, dass die verschiedenen Funktionen des Systems gut integriert sind.	12345	
6. Ich finde, dass es im System zu viele Inkonsistenzen gibt.	12345	
7. Ich kann mir vorstellen, dass die meisten Leute das System schnell zu beherrschen lernen.	12345	
8. Ich empfinde die Bedienung als sehr umständlich.	12345	
9. Ich habe mich bei der Nutzung des Systems sehr sicher gefühlt.	12345	
10. Ich musste eine Menge Dinge lernen, bevor ich mit dem System arbeiten konnte.	12345	

Siehe Rückseite

11. Das System leistet echte Hilfestellung für bekannte Use-Cases oder Szenarien .

1	2	3	4	5

12. Das System gliedert sich gut in bestehende Infrastruktur ein

1	2	3	4	5

13. Das System stellt einen Mehrwert dar gegenüber bisher genutzten Tools/Systemen.

1	2	3	4	5

14. Die vom System bereitgestellten Daten und Darstellungen, sind klar und schnell verständlich.

1	2	3	4	5

15. Das System läuft flüssig und ist ohne Störungen.

1	2	3	4	5

16. Folgende Funktionen fehlen noch:

.....

17. Weitere Anmerkungen:

.....

.....

.....

.....

Evaluation Tool "Kronos"

Stimme zu

Stimme nicht zu

1. Ich kann mir sehr gut vorstellen, das System regelmäßig zu nutzen.

X				
1	2	3	4	5

2. Ich empfinde das System als unnötig komplex.

				X
1	2	3	4	5

3. Ich empfinde das System als einfach zu nutzen.

X				
1	2	3	4	5

4. Ich denke, dass ich technischen Support brauchen würde, um das System zu nutzen.

			X	
1	2	3	4	5

5. Ich finde, dass die verschiedenen Funktionen des Systems gut integriert sind.

	X			
1	2	3	4	5

6. Ich finde, dass es im System zu viele Inkonsistenzen gibt.

				X
1	2	3	4	5

7. Ich kann mir vorstellen, dass die meisten Leute das System schnell zu beherrschen lernen.

X				
1	2	3	4	5

8. Ich empfinde die Bedienung als sehr umständlich.

				X
1	2	3	4	5

9. Ich habe mich bei der Nutzung des Systems sehr sicher gefühlt.

X				
1	2	3	4	5

10. Ich musste eine Menge Dinge lernen, bevor ich mit dem System arbeiten konnte.

				X
1	2	3	4	5

11. Das System leistet echte Hilfestellung für bekannte Use-Cases oder Szenarien.

u
✓

X				
1	2	3	4	5

12. Das System gliedert sich gut in bestehende Infrastruktur ein

X				
1	2	3	4	5

13. Das System stellt einen Mehrwert dar gegenüber bisher genutzten Tools/Systemen.

X				
1	2	3	4	5

14. Die vom System bereitgestellten Daten und Darstellungen, sind klar und schnell verständlich.

	X			
1	2	3	4	5

15. Das System läuft flüssig und ist ohne Störungen.

	X			
1	2	3	4	5

16. Folgende Funktionen fehlen noch:

Tags: stat. Uses die Daten einzuschränken wäre dies via tags hilfreich insb. dass die tag ebenfalls in der URL festgehalten werden

17. Weitere Anmerkungen:

"Kanos" ist ein Tool welches einen echten Mehrwert für uns generiert, tägliche Abläufe vereinfacht und durch uns höchstwahrscheinlich auch weiterentwickelt wird. Vielen Dank!

Evaluation Tool "Kronos"

	Stimme zu	Stimme nicht zu											
1. Ich kann mir sehr gut vorstellen, das System regelmäßig zu nutzen.	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 20%; text-align: center;">X</td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">2</td> <td style="text-align: center;">3</td> <td style="text-align: center;">4</td> <td style="text-align: center;">5</td> </tr> </table>	X					1	2	3	4	5		1
X													
1	2	3	4	5									
2. Ich empfinde das System als unnötig komplex.	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%; text-align: center;">X</td> <td style="width: 20%;"></td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">2</td> <td style="text-align: center;">3</td> <td style="text-align: center;">4</td> <td style="text-align: center;">5</td> </tr> </table>				X		1	2	3	4	5		4
			X										
1	2	3	4	5									
3. Ich empfinde das System als einfach zu nutzen.	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 20%; text-align: center;">X</td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">2</td> <td style="text-align: center;">3</td> <td style="text-align: center;">4</td> <td style="text-align: center;">5</td> </tr> </table>	X					1	2	3	4	5		1
X													
1	2	3	4	5									
4. Ich denke, dass ich technischen Support brauchen würde, um das System zu nutzen.	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%; text-align: center;">X</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">2</td> <td style="text-align: center;">3</td> <td style="text-align: center;">4</td> <td style="text-align: center;">5</td> </tr> </table>					X	1	2	3	4	5		5
				X									
1	2	3	4	5									
5. Ich finde, dass die verschiedenen Funktionen des Systems gut integriert sind.	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 20%;"></td> <td style="width: 20%; text-align: center;">X</td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">2</td> <td style="text-align: center;">3</td> <td style="text-align: center;">4</td> <td style="text-align: center;">5</td> </tr> </table>		X				1	2	3	4	5		2
	X												
1	2	3	4	5									
6. Ich finde, dass es im System zu viele Inkonsistenzen gibt.	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%; text-align: center;">X</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">2</td> <td style="text-align: center;">3</td> <td style="text-align: center;">4</td> <td style="text-align: center;">5</td> </tr> </table>					X	1	2	3	4	5		5
				X									
1	2	3	4	5									
7. Ich kann mir vorstellen, dass die meisten Leute das System schnell zu beherrschen lernen.	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 20%; text-align: center;">X</td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">2</td> <td style="text-align: center;">3</td> <td style="text-align: center;">4</td> <td style="text-align: center;">5</td> </tr> </table>	X					1	2	3	4	5		1
X													
1	2	3	4	5									
8. Ich empfinde die Bedienung als sehr umständlich.	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%; text-align: center;">X</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">2</td> <td style="text-align: center;">3</td> <td style="text-align: center;">4</td> <td style="text-align: center;">5</td> </tr> </table>					X	1	2	3	4	5		5
				X									
1	2	3	4	5									
9. Ich habe mich bei der Nutzung des Systems sehr sicher gefühlt.	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%; text-align: center;">X</td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">2</td> <td style="text-align: center;">3</td> <td style="text-align: center;">4</td> <td style="text-align: center;">5</td> </tr> </table>			X			1	2	3	4	5		3
		X											
1	2	3	4	5									
10. Ich musste eine Menge Dinge lernen, bevor ich mit dem System arbeiten konnte.	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%; text-align: center;">X</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">2</td> <td style="text-align: center;">3</td> <td style="text-align: center;">4</td> <td style="text-align: center;">5</td> </tr> </table>					X	1	2	3	4	5		5
				X									
1	2	3	4	5									

32

Siehe Rückseite

SUS 80

11. Das System leistet echte Hilfestellung für bekannte Use-Cases oder Szenarien.

X				
1	2	3	4	5

12. Das System gliedert sich gut in bestehende Infrastruktur ein

X				
1	2	3	4	5

13. Das System stellt einen Mehrwert dar gegenüber bisher genutzten Tools/Systemen.

X				
1	2	3	4	5

14. Die vom System bereitgestellten Daten und Darstellungen, sind klar und schnell verständlich.

	X			
1	2	3	4	5

15. Das System läuft flüssig und ist ohne Störungen.

	X			
1	2	3	4	5

16. Folgende Funktionen fehlen noch:

Leid wäre wenn man Fälle benennen & unterscheiden könnte. ☒

17. Weitere Anmerkungen:

Danke für eure Unterstützung & Fehlerberichte. War sehr angenehm.

Die Angabe in Byte ist teilweise etwas unständlich. Angabe in KB, MB, GB wäre toll.

☒ z.B. statt über User über Tags direkt, so müsste Cache nicht gelöscht & neu eingelegt werden.

Evaluation Tool "Kronos"

	Stimme zu	Stimme nicht zu
1. Ich kann mir sehr gut vorstellen, das System regelmäßig zu nutzen.	X 12345	1
2. Ich empfinde das System als unnötig komplex.	 X 12345	5
3. Ich empfinde das System als einfach zu nutzen.	X 12345	1
4. Ich denke, dass ich technischen Support brauchen würde, um das System zu nutzen.	 X 12345	5
5. Ich finde, dass die verschiedenen Funktionen des Systems gut integriert sind.	X 12345	1
6. Ich finde, dass es im System zu viele Inkonsistenzen gibt.	 X 12345	5
7. Ich kann mir vorstellen, dass die meisten Leute das System schnell zu beherrschen lernen.	X 12345	1
8. Ich empfinde die Bedienung als sehr umständlich.	 X 12345	5
9. Ich habe mich bei der Nutzung des Systems sehr sicher gefühlt.	X 12345	1
10. Ich musste eine Menge Dinge lernen, bevor ich mit dem System arbeiten konnte.	 X 12345	5

30

Siehe Rückseite

Sus 75

11. Das System leistet echte Hilfestellung für bekannte Use-Cases oder Szenarien .

X				
1	2	3	4	5

12. Das System gliedert sich gut in bestehende Infrastruktur ein

	X			
1	2	3	4	5

13. Das System stellt einen Mehrwert dar gegenüber bisher genutzten Tools/Systemen.

X				
1	2	3	4	5

14. Die vom System bereitgestellten Daten und Darstellungen, sind klar und schnell verständlich.

	X			
1	2	3	4	5

15. Das System läuft flüssig und ist ohne Störungen.

	X			
1	2	3	4	5

16. Folgende Funktionen fehlen noch:

.....

17. Weitere Anmerkungen:

.....

.....

.....

.....