DokterService
A Security Information and Event Management System (SIEMS)
W. van der Lee
M. el Maouchi
A.C.A. Mauricio
R.A. Sewsingh
DokterService

A Security Information and Event Management System (SIEMS)

by

W. van der Lee
M. el Maouchi
A.C.A. Mauricio
R.A. Sewsingh

in partial fulfillment of the requirements for the degree of

Bachelor of Science
in Computer Science and Engineering

at the Delft University of Technology.

Coaches:
Dr. C. Doerr, TU Delft
M. Hovens, KPMG IT Advisory
D. Wagenaar, KPMG IT Advisory

Other committee members:
F. Hermans, TU Delft
M. Larson, TU Delft

This thesis is confidential and cannot be made public until June 19, 2015.

An electronic version of this thesis is available at http://repository.tudelft.nl/.
This report is submitted in partial fulfillment of the requirement for the degree of Bachelor in Computer Science & Engineering at the University of Delft and concludes the Bachelor's Final Year Project. The project endured from April 2015 to June 2015 and mainly took place in KPMG's office in Amstelveen, the Netherlands.

We would like to thank KPMG IT Advisory for giving us the opportunity to fulfill our final year project. We moreover would like to thank Max Hovens, Daan Wagenaar, Jeroen de Wit and Erwin Hansen for their support and guidance during the project. They have provided us with lots of trust and flexibility during this project and it has been an amazing experience for us working with them. We also would like to thank Ruud Verbij for putting us in contact with the KPMG IT Advisory division and directing us to the right people for creating such a wonderful project. We would also like to thank our coach, Dr. Christian Doerr, for coaching us and giving us clear and detailed instructions when needed.

W. van der Lee
M. el Maouachi
A.C.A. Mauricio
R.A. Sewsingh
Delft, July 2015
# Contents

**List of Figures** vi

**List of Tables** vii

1 **Introduction** 1

2 **Problem Analysis** 2
   2.1 Customer's Profile. 2
   2.2 Existing Proof of Concept 2
   2.3 Requirements 3
      2.3.1 MoSCoW-model 4
      2.3.2 Functional vs Non-Functional 4

3 **Research Document** 5
   3.1 Data storage 5
      3.1.1 MongoDB 5
   3.2 Invalidating Existing Proof of Concept 6
      3.2.1 Proof of Concept 6
   3.3 Network Tools 7
      3.3.1 Suricata 7
      3.3.2 Oinkmaster 7
      3.3.3 Bro 7
      3.3.4 RESTful Web Service 7
   3.4 Analysis 9
   3.5 Solution Sketch 10
   3.6 General Document 10
   3.7 Miscellaneous External Scripts and Libraries 10
      3.7.1 Charts 11
      3.7.2 Mean.js 11
      3.7.3 FileSaver 11

4 **Project Strategy** 12
   4.1 Teamwork 12
   4.2 Software Development Strategy 12
      4.2.1 Scrum 12
      4.2.2 Trello 14
      4.2.3 Revision Control with Git 14
      4.2.4 Code Style 15
      4.2.5 Product Timeline and Tasks 15

5 **Process Reflection** 18
   5.1 Reflection on Software Development Strategy 18
   5.2 Demo 18
   5.3 Change of the requirements 19

6 **Product** 21
   6.1 User Interface 21
      6.1.1 Login Screen 21
      6.1.2 Run Services 21
      6.1.3 Generate Reports 21
   6.2 Web Routing 22
      6.2.1 Sitebuilding 25
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.3</td>
<td>IDS Services</td>
<td>25</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Directories</td>
<td>25</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Managers</td>
<td>26</td>
</tr>
<tr>
<td>6.3.3</td>
<td>NetworkTools</td>
<td>26</td>
</tr>
<tr>
<td>6.3.4</td>
<td>Parsing Output</td>
<td>26</td>
</tr>
<tr>
<td>6.3.5</td>
<td>Exporting Parse Results</td>
<td>27</td>
</tr>
<tr>
<td>6.3.6</td>
<td>Observer Pattern</td>
<td>27</td>
</tr>
<tr>
<td>6.4</td>
<td>Database</td>
<td>27</td>
</tr>
<tr>
<td>6.4.1</td>
<td>Database structure</td>
<td>28</td>
</tr>
<tr>
<td>6.4.2</td>
<td>Query</td>
<td>29</td>
</tr>
<tr>
<td>6.5</td>
<td>Report</td>
<td>29</td>
</tr>
<tr>
<td>6.5.1</td>
<td>Document Creation</td>
<td>29</td>
</tr>
<tr>
<td>6.5.2</td>
<td>Different types of Reports</td>
<td>30</td>
</tr>
<tr>
<td>6.5.3</td>
<td>Whitelist</td>
<td>31</td>
</tr>
<tr>
<td>6.5.4</td>
<td>PCAP Information</td>
<td>31</td>
</tr>
<tr>
<td>6.6</td>
<td>Network Analysis</td>
<td>31</td>
</tr>
<tr>
<td>6.6.1</td>
<td>Analysis</td>
<td>31</td>
</tr>
<tr>
<td>6.6.2</td>
<td>Information Gathering</td>
<td>32</td>
</tr>
<tr>
<td>6.6.3</td>
<td>Generating the Document</td>
<td>32</td>
</tr>
<tr>
<td>6.7</td>
<td>Settings</td>
<td>32</td>
</tr>
<tr>
<td>6.7.1</td>
<td>SettingProcessor</td>
<td>32</td>
</tr>
<tr>
<td>6.7.2</td>
<td>System Configurations</td>
<td>32</td>
</tr>
<tr>
<td>6.7.3</td>
<td>Report Configurations</td>
<td>33</td>
</tr>
<tr>
<td>6.8</td>
<td>Extensibility</td>
<td>33</td>
</tr>
<tr>
<td>6.8.1</td>
<td>IDS Service Extensibility</td>
<td>33</td>
</tr>
<tr>
<td>6.8.2</td>
<td>Parsing Extensibility</td>
<td>34</td>
</tr>
<tr>
<td>7</td>
<td>Validation</td>
<td>35</td>
</tr>
<tr>
<td>7.1</td>
<td>Requirement Validation</td>
<td>35</td>
</tr>
<tr>
<td>7.1.1</td>
<td>Requirements</td>
<td>35</td>
</tr>
<tr>
<td>7.1.2</td>
<td>Conclusion</td>
<td>38</td>
</tr>
<tr>
<td>7.2</td>
<td>Future Work Recommendations</td>
<td>38</td>
</tr>
<tr>
<td>8</td>
<td>Quality Assurance</td>
<td>39</td>
</tr>
<tr>
<td>8.1</td>
<td>Testing</td>
<td>39</td>
</tr>
<tr>
<td>8.1.1</td>
<td>Integration of testing</td>
<td>40</td>
</tr>
<tr>
<td>8.1.2</td>
<td>Types of tests</td>
<td>40</td>
</tr>
<tr>
<td>8.1.3</td>
<td>Test frameworks</td>
<td>40</td>
</tr>
<tr>
<td>8.2</td>
<td>CodePro AnalytiX</td>
<td>41</td>
</tr>
<tr>
<td>8.2.1</td>
<td>SIG</td>
<td>41</td>
</tr>
<tr>
<td>8.3</td>
<td>Integration</td>
<td>43</td>
</tr>
<tr>
<td>9</td>
<td>Conclusion</td>
<td>45</td>
</tr>
<tr>
<td>A</td>
<td>SIG Evaluation 1</td>
<td>46</td>
</tr>
<tr>
<td>A.1</td>
<td>Dutch</td>
<td>46</td>
</tr>
<tr>
<td>A.2</td>
<td>English</td>
<td>47</td>
</tr>
<tr>
<td>B</td>
<td>SIG Evaluation 2</td>
<td>48</td>
</tr>
<tr>
<td>B.1</td>
<td>Dutch</td>
<td>48</td>
</tr>
<tr>
<td>B.2</td>
<td>English</td>
<td>48</td>
</tr>
<tr>
<td>C</td>
<td>Component Architecture</td>
<td>49</td>
</tr>
<tr>
<td>D</td>
<td>Bro and Suricata output fields</td>
<td>55</td>
</tr>
<tr>
<td>E</td>
<td>Info Sheet</td>
<td>59</td>
</tr>
<tr>
<td>F</td>
<td>Final Requirements</td>
<td>61</td>
</tr>
<tr>
<td>G</td>
<td>Final Timeline</td>
<td>63</td>
</tr>
</tbody>
</table>
Bibliography 64
List of Figures

3.1 MongoDB Structure Diagram ................................................................. 6
3.2 Pre Elementary Sketch of the DokterService Structure ............................. 10
4.1 Software Component Architecture on Settings ...................................... 17
5.1 Snippet of our Product Backlog in iteration #8 ...................................... 19
6.1 Example of a SessionID cookie and it’s contents .................................... 24
6.2 Web Navigation relationships ............................................................... 24
6.3 Code snippet from template.html ......................................................... 25
6.4 The Implemented Observer Pattern ...................................................... 28
6.5 Abstract Methods from NetworkTool: The IDS Dependable Functions ........ 34
8.1 Example CodePro AnalytiX Metrics Analysis ......................................... 42
C.1 Software Component Architecture on IDS Services ................................. 50
C.2 Software Component Architecture on Settings ....................................... 51
C.3 Software Component Architecture on Web Routing ................................. 52
C.4 Network Analysis Diagram ................................................................. 53
C.5 Database Structure Diagram ............................................................... 54
LIST OF TABLES

2.1 Classification of the requirements according to the MoSCoW-model. .............................. 4
2.2 Functional- and Non-Functional classification of the requirements. ............................... 4

4.1 Initial Project Timeline. ........................................................................................................ 15

6.1 List of *Rest-classes and their associated URLs ............................................................. 23
6.2 Summary of Interconnected Observe Instances ................................................................. 27

D.1 Suricata fields ..................................................................................................................... 55
D.2 Bro fields ............................................................................................................................ 56
D.3 Fields shown in the settings-section of the General Report. ........................................... 57
D.4 Fields used for the Malicious Report. ............................................................................... 58

G.1 Final Project Timeline. ....................................................................................................... 63
Companies all over the world are adapting exhaustively to the new digital era. New buzz topics as Cloud Computing and Big Data Analytics are being implemented to both their internal and external organization. Employees of a company are for example communicating by e-mail and sensitive data is often shared over a network or other IT-resources. Cyber threats are among the rise of the digital adaptation. As more companies integrate digital services, the organization's IT infrastructure becomes larger and more complex. The infrastructure could possibly be vulnerable to information breaches due to unwanted network activities.

To determine whether a company may unknowingly be a victim of cyber crime, one could analyze the company's network traffic with a Network Intrusion Detection Systems (NIDS). KPMG's Information Protection Services unit wanted a custom made Security Information and Event Management System (SIEMS) in order to manage these NIDSs, hence project DokterService arose. The purpose of the DokterService is to analyze captured network traffic and output a network analysis report which represents the "medical state" of a company's network infrastructure. The aim of our final project was to create such a tool. A proof of concept had already been created by KPMG, by means of a incoherent set of tools. This existing proof of concept will briefly be discussed in chapter 2. In order to develop the DokterService as one automated tool, the product needs to be redesigned, restructured and reimplemented.

This document describes the full-stack development process of the DokterService, including a research section, code quality feedback from Software Improvement Group (SIG) and a precise description about our developed solution to KPMG's problem. Chapter 2 contains a detailed problem analysis and states the initial product requirements that have been defined together with our stakeholders. Chapter 3 contains our research section, where our related research results are shown and discussed. Chapter 4 describes our project strategy, which will discuss our strategies for teamwork, software development and the product timeline. Chapter 5 will reflect on the aforementioned process strategies. This chapter will among others validate the way how we have applied scrum during the project. Chapter 6 gives an accurate outline of the different service components of our software product, as well as a description of the product's information flow throughout our product. Chapter 7 is dedicated to the requirement validation. Our stakeholder's final feedback will be addressed in this chapter as well. Chapter 8 contains information about the quality assurance of our software product. Central topics in this chapter are testing and code quality. The section about code quality discusses the feedback we have received from SIG and other analytical tools. Finally chapter 9 states our final conclusion.
This chapter decomposes the needs of the customer into clear requirements. In section 2.1 we will analyze the customer's profile and needs. Section 2.2 discusses the existing proof of concept KPMG already developed. Section 2.3 states the requirements that have been defined and agreed upon by us together with our stakeholders and TU coach prior to the software development.

2.1. Customer’s Profile
KPMG is a multinational firm, which is focused on three pillars: tax advisory, accountancy and advisory. The project is proposed by the advisory pillar, specifically IT Advisory within Risk Consulting from the unit Information Protection Services (IPS). IPS is specialized in all sorts of IT Security related assignments from their clients. IPS performs technical penetration tests to large organizational cyber transformations, from IT audits to code reviews and from drafting mobile device management policies to drafting Enterprise risk management frameworks.

This project arose in the context of IT audit, where IT infrastructures and processes are being analyzed. Among the analysis, one focus could be network traffic, as the traffic tells a lot of the internal processes. KPMG wants a tool that fully analyzes recorded network traffic, and outputs a report containing the "medical" condition of the company of which the recorded network belongs to. The goal of this project is to design and develop a "Security Information and Event Management System" that manages Network Intrusion Detection services and provides a report specifying the condition of the analyzed network.

2.2. Existing Proof of Concept
The above specified services were already deployed by KPMG. However, these services were a collection of incoherent subservices. The subservices were executed from the command line. The report was a simple interactive dashboard based on all produced log-files. The service in general can be described as follows:

1. KPMG records network traffic at a company by connecting a “Micro-Instance” to their network. A Micro-Instance is a little box, which is able to capture and record network traffic. The recorded network is stored in files storing the captured traffic in so called .pcap-files that is composed of low-level packet captures. The duration of the network capture may vary from one day to two weeks.

2. After capturing the network traffic, the “Micro-Instance” is taken back to KPMG where the .pcap-files will be further analyzed on their own server, the DokterService-Server.

3. NIDSs Bro 3.3.3 and Suricata 3.3.1 analyze the .pcap-files. They each produce log-files.

4. The log-files are viewed by a combination of tools called the ELK Stack: ElasticSearch, Logstash and Kibana. Logstash is used to parse the log-files. Elasticsearch is used as an elastic database where the parsed log-files are stored. Kibana is used as a data-visualization user interface, to analyze the contents of ElasticSearch [1].

The largest problem of the above specified services, is that they have to be initiated and maintained by the command-line interface of the DokterService-Server. Another problem KPMG had encountered with
their existing proof of concept is that it was not build in a modular way. Extending the DokterService with additional intrusion detection tools, is a very real and possible course of events. Extending the existing proof of concept was hard, due to the data was not maintained properly. For example, the generated output from Bro and Suricata in the existing proof of concept, could be distinctly analyzed, but never matched. The latter was due to the inconsistencies in timestamp format and output from both tools. The generated data was also tremendously cluttered, because the log-files provided no more info to Logstash, Elasticsearch and Kibana, except from the generated output headers. The generated data also becomes cluttered because every single tool's output should be analyzed distinctive.

2.3. Requirements
Requirements for the analysis are defined as follows:

1. Data input consists of multiple .pcap-files that contains captured network traffic.
2. .pcap-files are parsed with the original timestamps.
3. The captured data is transferred from the micro instance to the analysis instance by means of an automated system that is user initiated.
4. The information stored in a database should be an application independent format.
5. The database should scale well for large amounts of data (> 10 GB).
6. The application should be built in a modular way, allowing addition and removal of modules. (e.g. different modules for the different functionality, extra NIDSs).
7. Computation time should not exceed 2 days (48 hours).
8. The application is web-based. No command line interaction should be necessary to upload a .pcap-file, generate a report or view any retrieved data.
9. The access to the web-based application is properly authenticated.
10. Malware analysis is done with at least:
    (a) Bro
    (b) Suricata
11. The rule list is updated prior to the start of an analysis.
12. Automatic file carving (i.e potential malicious files are carved from the traffic and stored on disc)(Optional)
14. The analysis report should contain:
    (a) Top 10 lists of:
        i. Most visited countries
        ii. Most visited IPs
        iii. Most used protocols
        iv. Most used ports
        v. Most origination IPs
        vi. Most visited URLs
        vii. Largest files downloaded
        viii. Most used User Agents (HTTP(S) traffic)
    (b) List of malware that was found (based on Suricata output; e.g. Critical / High alerts)
    (c) Analysis of bandwidth usage (optional).
15. The output format of the report is PDF in a pre-defined template. Preferably also a format which is editable (.doc(x) or otherwise editable is fine)
2.3.1. **MoSCoW-model**

Prior to the design and development of the product and product planning, further analysis on the requirements is needed. Our first step to analyze the requirements is grading the requirements according to the MoSCoW model. The MoSCoW model is a prioritization technique used to specify the importance of the requirements, according to the necessity of their implementation. The requirements can be classified into one of four possible classifications in the MoSCoW model which run from a high priority to a low priority. A requirement is either a "Must have", a "Should have", a "Could have" and a "Won't have". The reason to state Won't have -requirements is that this will not be developed during this project, but that it could probably be requested in the future. The probable future extension of the DokterService is at this point proper information, because the requirement can then be taken into account while designing and implementing the whole software product. Next you will find the requirements classified according to the MoSCoW-model.

<table>
<thead>
<tr>
<th>Classification</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Must Have</td>
<td>1, 4, 6, 7, 8, 10, 14a, 14b</td>
</tr>
<tr>
<td>Should Have</td>
<td>2, 3, 5, 9, 11, 15</td>
</tr>
<tr>
<td>Could Have</td>
<td>12, 13, 14c</td>
</tr>
<tr>
<td>Won't Have</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.1: Classification of the requirements according to the MoSCoW-model.

2.3.2. **Functional vs Non-Functional**

As the MoSCoW-model states above, requirements can be subdivided into functional or non-functional requirements. There are different reasons to subdivide the requirements into functional and non-functional categories. The main reason is to clear the responsibility for implementation. Especially non-functional performance-related requirements have to be taken into account while allocating requirements. For instance one could impossibly implement a certain performance-related requirement in a software product, when the main problem is the hardware configuration. Therefore in order to address the responsibility of the requirements, we also classify them as functional and non-functional requirements.

<table>
<thead>
<tr>
<th>Classification</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional</td>
<td>1, 2, 3, 8, 9, 12, 13, 14, 15</td>
</tr>
<tr>
<td>Non-Functional</td>
<td>4, 5, 6, 7, 10, 11</td>
</tr>
</tbody>
</table>

Table 2.2: Functional- and Non-Functional classification of the requirements.
This chapter describes our research findings. It contains a summary of the results we have found during the research phase of the project.

3.1. DATA STORAGE

Data storage is an essential part of a system. Permanent storage can be done either by file system storage or database storage. Our goal is to analyze data, so searching by use of indices is required. Therefore the data will be stored in a database, which can provide us this requirement.

Next we had to consider what kind of database we wanted to use. From our university's prerequisites, we all had advanced knowledge of SQL-syntax-based databases. SQL-syntaxed databases, or relational databases, like MySQL, SQLite and PostgreSQL are known to be less scalable and provide inferior performance in comparison with NoSQL databases. From our problem analysis in section 2.3 it purified that our input file could extend in size to even 1TB of data. It became clear we had to use a NoSQL database, which means we need scalability and superior performance provision.

We conclude with a list of advantages of a NoSQL database opposed to an SQL-database [2]. Opposed to a SQL-database, a NoSQL database is designed to:

- handle large volumes of structured, semi-structured and unstructured data;
- address agile sprints, quick iterations, and frequent code pushes;
- address object-oriented programming that is easy to use and flexible;
- be efficient, scale-out architecture instead of expensive, monolithic architecture.

This is exactly what we need, so the conclusion to use a NoSQL database was easily made.

Considering different NoSQL databases, we found that they are classified by their different storage data models [3]. With No-SQL databases information can be stored according to key-value matches, tuple store, object database etc.. Since we want to store and even analyze results of tools like Bro and Suricata, we preferably want to store information as a document, because Bro and Suricata represent their results in a document-based language. Moreover, they are able to output their results in a JSON-format, which can be converted to a document object, without information-loss, just by specifying all the keys and values per field. We then decided to use MongoDB, because it is a Document Stored database, which can process inputs in JSON-format.

3.1.1. MONGODB

MongoDB is a document-based NoSQL database. The key features of this database is its high performance, high availability and automatic scaling [4]. An overview of the structure of a MongoDB database can be seen in figure 3.1. Using a MongoDB client, one can create multiple databases [5]. Each database contains collections which can be seen as tables. An element in a collection is called a document, which is why MongoDB is a document-based database.
MongoDB has several drivers which support different kinds of environments. For example: Python, Node.JS, C++, C# and Java. Because every project member has advanced knowledge in programming in Java, we decided to use the Java driver. This was also one of the major reasons we decided to create our product in the Java programming language.

3.2. INVALIDATING EXISTING PROOF OF CONCEPT

KPMG had their own proof of concept which uses the Elasticsearch Logstash and Kibana (ELK) stack. What these are and why KPMG required to have it set up in a different way, is explained next.

- **Elasticsearch**
  Elasticsearch is built on top of the Apache Lucene framework and is a real-time distributed search and analytics engine. It provides a search engine with a RESTful web interface and schema-free JSON documents. Mega-corporations like Wikipedia, The Guardian, Stack Overflow and GitHub use Elasticsearch in different ways to their own needs [6].

- **Logstash**
  Logstash is a tool for processing and managing events and logs. It provides a pipeline for storing, querying and analyzing log files, so you can use it to collect, parse and store logs for further use [7].

- **Kibana**
  Kibana is a visualization platform that is designed to work with Elasticsearch. You can use Kibana to easily understand large volumes of data. Data can be visualized in a variety of charts, tables and maps [8].

3.2.1. PROOF OF CONCEPT

Elasticsearch can be used as a back-end data store for Logstash and this can be combined with Kibana as the front-end to visualize and report the data. This is exactly what KPMG has as their proof of concept, but they are experiencing problems with this. The ELK stack limits what they can do with the output. You cannot define your own output and it is also not possible to group and merge separate outputs with each other, which means that it is very difficult to link certain data with each other. This made it difficult for KPMG to do their own analysis and means that they are limited to what the ELK stack outputs. Because of this they were generating false-positives and true-negatives, which is one tries to avoid because this increases the chances of missing something essential in the network.
3.3. NETWORK TOOLS

This subsection describes which tools are being used by our solution, in order to perform a network analysis. In section 3.3.1 we will discuss our research findings on the Network Intrusion Detection System Suricata. In section 3.3.3 we will discuss why we integrated the Bro framework.

3.3.1. SURICATA

As told in the beginning of this section, Suricata is a Network Intrusion Detection System (NIDS). Besides that, Suricata is also an Intrusion Prevention System (IPS) and a Network Security Monitoring Engine (NSME). We will be using Suricata mainly because of the network intrusion detection facilities. The reason why we integrated Suricata in our solution is because it is one of the requirements from the stakeholders. To address the customer’s motivation, Suricata has a lot of advantages which is able to meet certain demands regarding the advanced network analysis. In our research about Suricata we found that Suricata can work with very large volumes of network traffic packages, the so called PCAP-files. This is of importance to our project because of the large size of PCAP-files that can be uploaded and the time constraint given by the requirements (See section 2.3). Suricata can also do protocol inspection, so one does not have to rely on port numbers to identify traffic. One can then easily examine HTTP traffic no matter what port it is on. It is also possible to look inside of the protocol streams and extract the files, so they can then be examined, and one can examine TSL/SSL certificates and match them based on their fingerprints, so even if one cannot intercept the TLS/SSL connection, one can still exert some control over it. Suricata has the ability to understand level 7 of the OSI model, which enhances its ability of detecting malwares. We consider to use all of these features Suricata enables us to use.

Suricata's attempt to detect malware from PCAP-files, is by matching the content against a set of rules. These rules are specific strings of malicious network traffic. Because malware always changes as new viruses and malicious activities arise, these rules need to be updated as well. These rules are among others daily updated at www.emergingthreats.net. Because our solution needs to find the most recent threats from the PCAP-files, we will update the set of rules every time the DokterService is started. Updating these rules will be done with a tool called Oinkmaster, which will be further elaborated on in section 3.3.2.

3.3.2. OINKMASTER

As told in the previous section about Suricata, there needs to be some rule management, regarding the ruleset of malicious threats. Requirement 11 of our stakeholders states that this list needs to be updated prior to the start of an analysis. Hence we want to integrate the update of the rules to be done automatically, prior to the start of the run of Suricata. To manage and update these rules we have chosen to use Oinkmaster. Oinkmaster is a script that will help you update and manage your rules. New sets of rules will be downloaded from the URL: https://rules.emergingthreats.net/open/suricata/rules/, which will then be used to update the local rules.

3.3.3. BRO

Another NIDS we want to implement in our product is Bro. Requirement 10a from our stakeholders specifies that Bro should be used for the analysis. For our project, not only do we need to detect malware, we also need to track suspicious traffic. This is where Bro comes in handy. It can be used to reduce incoming packet streams into higher-level events and we are able to apply our own customizable scripts to determine the necessary course of actions. This last part is useful, since we additionally are going to implement our own analysis on the network traffic to detect suspicious events.

3.3.4. RESTful WEB SERVICE

According to requirement 8 it is required to have a web-interface where the user can log into and use the system. This web-interface should be properly authenticated according to requirement 9. To achieve this there is a need to have communication between the backbone, written in Java, and the web-interface. After a period of research it became clear that this is easily done by means of a RESTful Service. The only communication between the backbone and the web-interface should be done by the HTTP-protocol. In this subsection we will discuss the two frameworks that were found to achieve the solution for this problem. This will be primarily about the backbone. Besides the backbone, this subsection will discuss the web-interface and the manner that was set up.
THE BACKBONE

First of all, to be able to provide a communication between a system written in a native language and a web-interface using HTTP-requests, a server is required. Currently, there are two major frameworks who provide the ability to set up a server using Java. One of the frameworks is called Springboot [9]. This framework does not exist for a long period of time, but has matured enormously throughout the years. There was sufficient documentation and there was enough information available online. However, there was a limitation that troubled is. This limitation was of importance for the primary design of the product. It was the fact that it was not possible to start a server and listen to a various amount of ports. A server creation with the Spring-framework only has the ability to listen to one port at a time. When other ports are needed, another server has to be started. This was the major limitation to make a switch to another framework, namely: Grizzly. Grizzly [10] is almost the same as Springboot. Most of the functionalities are the same, however syntaxes and the use of them are slightly different. A very useful and easy part of the framework, is the use of annotations. They are clear and very well documented. Our TU-coach also had recommended the use of Grizzly. It was easier to use and not so heavy according to him. This was another reason to make the switch to this framework. After switching the product to be able to work with Grizzly, we tried to make use of one server and different ports. However, to our surprise, this framework did not have that support either. This resulted in making a slight change in the product design. We kept using Grizzly since there was no reason to switch back to Springboot, which had the same functionality as Grizzly that was required.

With the functionality of setting a server up and making use of HTTP-requests, another framework was required to process the HTTP-requests by Java and produce output for the front-end. The most forward solution that was found is the use of the framework named Jersey [11]. Jersey has the ability to work seamlessly with the Grizzly-framework. It provides multiple methods to generate output to the front-end and GET-and POST-requests are handled easily. This eventually made the distinction between the functionality that requires either computing power or components of the system and the web-interface which contains HTML, CSS and Javascript. The next subsection discusses "Websockets".

WEB SOCKETS

The previously discussed Grizzly-framework has one big drawback: the server can only communicate to the end-user's browser, through HTTP-responses. The server might want to update a webpage, even when the end-user does not invoke a HTTP-request. In order to exemplify the latter case, we present you the scenario when a user has started the IDS services from the DokterService. The server then could be busy for a long period of time, even extending multiple days of computation. The end-user obviously wants to know the progress of the server or read the error messages (if any). In order to be able to see these logs, it is very tiresome for the user to send a new HTTP-request from the user's browser, in order to refresh or see these logs. Hence our server needs to be able to communicate to the end-user's browser on additional way, other than HTTP-requests. This is where Websockets come in handy.

We wanted to use Websockets for non-HTTP-request traffic. On the exact moment a user requests a valid webpage where non-HTTP-request traffic a Websocket could be instantiated from our server to the client's browser. We would then use Java's Websocket API for creating a server side Websocket. In the client's browser, we would use JavaScript to instantiate the connection to the Websocket on client side. Javascript has a Websocket Interface available at all times.

WEB INTERFACE

The user interface of our product should be web-based according to requirement 8 and properly authenticated according to requirement 9. Based on the implementation of the backbone the choice was made to use HTML, CSS and Javascript for the web-interface. The main reason why the web-interface is composed with only these three languages is because every team member is sufficiently proficient in any of these language and because these languages are compiled in the clients web-browser. The latter limits the communication with the Jersey server to a minimum, thus minimizing computation time of our product.

With regards to the requirement about proper authentication (requirement 9) we decided to set up the authentication as follow: The server will respond with services by reacting at GET- and POST-request fired at several paths our server listens to. All these paths need to be properly authenticated, by means of a logging-in system. After a discussion with our stakeholders regarding the level of security, the authentication should not be impenetrable, a simple login form which prompts the user for a username and password should suffice. With this in mind, a user-model should be taken into account while developing our product. The following points were to be considered:
1. Users need to be able to be created and deleted. We will integrate these options in our general setting page.

2. Users need to be stored in a database. Since we already have a local database client MongoDB, we will create an extra database "Authentication" where the user credentials are stored. We do however not want this database to be visible and accessible to anyone except our DokterService application. We therefore need to secure this specific database and connect with proper credentials to this database.

3. We do not want to store plain passwords of users in the database, because then our application may become a breach for revealing real credential information of real users. We will make use of already existing encryption models to store password hashes in the database Authentication. To make brute force password retrieval more difficult, we will be hashing the user passwords with a randomly generated salt.

4. Valid user authentication will result in an expirable session, in which the user can use the DokterService. Sessions should expire by default after 30 minutes after logging into the DokterService, after logging out or after closing the browser.

5. Valid sessions will get a unique valid sessionID, which will be stored in the browser's cookie. This cookie will be communicated upon requests.

### 3.4. Analysis

For this project, apart from the requirements given by the stakeholders, we wanted to implement our own analysis in the product, using creativity and knowledge of professionals. We discussed with some professionals in the field of network analysis and created a list containing several items which could detect malicious data traffic. This list contains the following:

1. **Suspicious User Agents**
   Different people use different user agents. But in general people tend to use a user agent which is commonly used. This analysis checks how many times a user agent is used and shows suspicious user agents based on that fact. The result is filtered with user agents by choice.

2. **Suspicious Source IP's and amount of DNS Requests**
   Sometimes people with malicious intents try to send data and information through DNS requests. This analysis checks how many times a specific source has used DNS requests. The amount of these DNS requests can be suspicious.

3. **Suspicious Files**
   Files can have different sizes, but some files are suspicious if they have a certain extension and a lower or higher than average size. This analysis checks for certain extensions and their corresponding suspicious sizes. Also the extensions 'exe', 'docm' and 'jar' are dangerous extensions, because if a virus or malware is stored in a file, it is most likely to have one of these extensions.

4. **Suspicious HTTP POST Requests**
   HTTP Post requests contain a content length. If this does not match with the actual content length of the POST request, this can be malicious and is therefore suspicious. There are of course several reasons why these do not match, such as a situation where the data traffic was cut, while sending. But this does not mean this occurrence is not suspicious.

5. **Suspicious Data Traffic (Night)**
   During the night, most companies are closed. This means data traffic should be close to zero. If there is a lot of data traffic at night, it means there probably is an unauthorized operation ongoing. This can also be an employee working late, but it is suspicious.

6. **DNS Tunneling**
   DNS Tunneling is a way to hide ones traffic or to sneak into public hotspots protected by HTTP redirects only [12]. Someone utilizing this, could correspond with the fact that this person has something to hide. This is obviously suspicious.
7. Traffic on not predefined ports
   Certain types of traffic travels over predefined ports. For example, ports which are used for SSL traffic include: 261, 443, and 448 [13]. If this is not the case, something malicious might be happening.

8. SSL traffic without a valid certificate (handshake)
   SSL traffic is established using a certain procedure. This procedure contains a handshake using certificates [14]. If this certificate is not valid, the chance is high that the endpoint of this SSL connection is malicious. Therefore this should always be checked.

3.5. Solution Sketch
   When creating a first draft, we have created and propounded a simple sketch as can be seen in 3.2. We know one or more .pcap-files should be consumed by our product, and which should by analyzed by at least the network tools Bro and Suricata. Prior to the analysis of the results, the Bro- and Suricata-output should be stored in a database for multiple easier, uniform and extensible analysis. Easier and uniform because certain possibly malicious events and packets can be cumulatively constructed as general objects, where data from multiple tools are merged. The data in the database can next be analyzed after which a report will follow.

![Figure 3.2: Pre Elementary Sketch of the DokterService Structure](image)

3.6. General Document
   From requirement no. 4 can be derived that a general document must be created for all tools. Bro and Suricata all have different output, so one document is created with fields both from Bro and Suricata, but not all of them. Bro and Suricata’s output can be found in Appendix D. As one can see these fields differ greatly from each other, but they do share a common field. These fieldsets are used to merge the output from Bro with Suricata. Now the question is, which fields are important enough to put in the general document? We have several reasoning for this, namely:

   - All shared fields, because we can use these fieldsets to merge the different output from Bro and Suricata as mentioned before.
   - Fields that are needed for our General Report. (Requirement no. 14a)
   - Fields that are needed for our Malware Report (requirement no. 14b). These are mostly Suricata fields, but weird_name and weird_addl_info from Bro are also used here.
   - Fields that are needed for our Analysis Report. In other words, fields that are used by our own analysis.

   A lot of information has been left out by not including all of the fields, but these left out fields do not have any added value to our product. By leaving these fields out we can save space and increase our application’s processing speed. How the General Document looks like can be found in Appendix D.3.

3.7. Miscellaneous External Scripts and Libraries
   This section will describe what other external scripts (mostly Javascript-scripts) or libraries (Java-libraries) we will be using, other than any aforementioned scripts or libraries.
3.7.1. Charts
In order to create charts based on table data, we will be using an external script called D3.js [15]. This script makes it very easy to create a chart. The data for the charts can be in JSON-format. Since a chart always belongs to a table containing data, we will traverse the table contents, creating the dataset and next providing a very interactive chart. The charts are interactive because end-users can exclude data entries interactively in the chart, just by clicking them.

In order to be able to download a generated chart for report-purposes, we will be generating a .png-image from the chart, which then will be downloadable with FileSaver.js as will be discussed in section 3.7.3. To be able to create an image from the generated charts we will first parse the chart’s HTML and CSS to a HTML5 Canvas with html2canvas.js [16]. Next we will be invoking a canvas2image.js script for parsing the Canvas to a .png-image.

3.7.2. Mean.js
Mean.js is a full-stack JavaScript solution that helps you build fast, robust, and maintainable production web applications using MongoDB, Express, AngularJS, and Node.js. This solution has been viewed because this could be used to build the front-end of our application. Mean.js has a couple interesting advantages:

1. A Single language is used in the whole application
2. It supports the MVC (Model-View-Control) pattern
3. Data is transferred in JSON format
4. Node.js is a large module library

Therefore Mean.js caught our attention. We tried to implement this in our application, but this resulted in a failure. Since none of the project members had experience with any of the frameworks, the knowledge about this system had to be acquired. After two days of exploring this solution, we decided that understanding and using it would take too much time. A simplistic front-end suffices for our application, so the abilities of this solution is not needed.

3.7.3. FileSaver
We will be using an external Javascript script called FileSaver.js [18]. FileSaver.js is the solution to saving files on the client-side, and is perfect for webapps that need to generate files, or for saving sensitive information that shouldn’t be sent to an external server. We will be using the functionality to download specific files for the download of

- CSV-files of table contents generated in the reports,
- created charts of table data in .png format.
This chapter will discuss all of our process-related strategies. Section 4.1 describes our teamwork. Agreements regarding for example presence, the location to work at and the time to start working are stated here. Section 4.2 gives a detailed overview of our software development strategies. This section discusses among others how we plan to apply the Scrum software development methodology and how we plan to revision our software with Git.

4.1. Teamwork

The four of us will be working full-time on this project. We plan to meet each other every working day to work together on this project. We will be mainly working at the KPMG office in Amstelveen, where we can have close contact with our stakeholders to ask questions regarding this project, show demo’s etc.. Another advantage of working at KPMG is to utilize other (informational) resources as, for example, cybersecurity experts to provide extra information or other viewpoints to be applied while implementing the product.

Mutual communication in the development team will be held via group-meetings (daily Scrum meetings, Section 4.2.1), face-to-face, a special WhatsApp-group, e-mail and Git commit-messages (Section 4.2.3). Communication with the client will be held through e-mail, face-to-face, meetings and demos. Communication with the TU coach will be done by e-mail and meetings.

4.2. Software Development Strategy

When developing a software product it is of importance to do this in a structured way to avoid chaos and to support a quick resolution to unexpected challenges. When developing a product it is easy to lose overview of the project. For example, changes to the product can be badly documented, which can lead to bad consequences if a project member is not informed about these changes. Knowledge about tasks and requirements can also be lost, due to bad structure. A structured way of developing helps to give a good overview of the development, which results in a higher efficiency when developing the product. It doesn’t assure you that everything goes perfect, but it does help the team to react in a proper way when things do not go as planned due to unexpected changes. A software development methodology tries to create this structure. So using a such a methodology is recommended when developing a product.

Section 4.2.1 will elaborate on the applied software development methodology: Scrum. We will mainly expound why and how we applied Scrum in our software development process. At section 4.2.2 the usage of the project management tool Trello will be discussed. zoSection 4.2.3 will explain how we revisioned the developed software using Git.

4.2.1. Scrum

There are several software development methodologies which try to create structure into a project. These software development methodologies [19] have been discussed internally between the team members. Because all team members had mostly applied the Scrum methodology in their previous software development experiences, the team agreed to proceed with the Scrum methodology. Scrum is an agile development methodology or rather a framework to use when developing software. Applying scrum excels in use when
requirements are unknown, unknowable or changing. Soon at the beginning of the project, the team figured out that Scrum was a good choice. The requirements and solutions evolved frequently through time. The stakeholder either removed, changed or added new requirements to the list of requirements set up at the beginning of the project.

**Division of roles**

In essence, when using Scrum it is required to divide two roles internally in the team. These two roles are of importance to divide as soon as possible. These roles are: the Product Owner and the Scrum Master. The Product Owner, or PO, is the client’s representative. The PO defines the features of the product, prioritizes the features and decides release dates and is definitive and decisive about the content of the product. The PO is of great importance to the project and since Scrum is built upon a product backlog, the PO is also indispensable to the entire team. The team agreed that this role should be for Mourad. Aside from the Product Owner, the Scrum Master, or SM, was also of great importance. The SM is the keeper of the Scrum process and the facilitator. An important task for him is to help everybody with the process and remove impediments from the team. Another task is to communicate with the Product Owner and help him out to keep the backlog in perfect shape. This role was given by the team to Wesley with support of the PO if necessary. Since the team was not big, four persons only, both the Product Owner and the Scrum Master needed to be part of the development team too. All four members were developers and KPMG was the stakeholder.

**Estimation of the backlog**

After the entire team had a meeting with the stakeholder, certain requirements were set up. Based on these requirements the product backlog was set up. The features and elements from the requirements are still vague. Nobody of the team knows how they are going to be developed. Even though this is the case, high level estimates were established. The estimates were done in a very high level, top down manner. This not only makes it possible to break them up easier later on in smaller parts, but also gives the stakeholder a clearer view of what the team is doing and developing. These initial estimates were also communicated to the stakeholder. The communication was done primarily by the PO, but with presence of the entire team to get multiple points of view when there was feedback.

After the estimates were established, the product backlog got estimated in points. By using points as measurement instead of a time unit, the question for the estimation shifted. It went from ”how long will it take?” to ”how big is it?”. This point system was based on the first seven distinct Fibonacci numbers point $p \in \{1, 2, 3, 5, 8, 13, 21\}$. Initially it was agreed upon that 1 point equals 4 hours of work. This shifted slightly throughout the development, but never shifted enormous to change the amount of hours that is equal to 1 point. Throughout the process the backlog sized up and estimates were set better.

**Sprint planning**

During the sprint planning, which was an iterative process, the requirements and estimates for tasks were clarified. This could not be done perfectly at once, but therefor was an iterative process. Clarifying the requirements was of utmost importance. Only by clarifying each requirement and breaking it into smaller parts, the team gained a better understanding of how the requirement and its sub-parts should get developed. This was not done by one person, but by the entire team. With some elements user stories were thought off which gave more insight and clearance. After the requirements are more clarified, they were broken up into tasks and each task got an estimation in hours. The team discusses the tasks and gives their estimation on the point. In case an estimation for a task was more than one day of work, the task gets broken up into even smaller parts. The benefit of tasks that are estimated under one day is that they are either done or not after the day. This improves the estimation of the entire team. Furthermore, the project consisted of 10 weeks to be spend on the development. This made it inconvenient to have long sprints. So the team agreed to have sprints of one week only.

**Review, reflect and repeat**

After each sprint, the entire team holds a review in a form of a meeting. This meeting generally takes 15-20 minutes. By reviewing it allows the team members to present their achievements. Furthermore, the stakeholders are able to view the progress and provide valuable feedback. This feedback then gets taken into consideration during the meeting again for the further steps of the project. Following the meeting of the review there is a meeting which is focused on reflecting on the previous sprint. During this meeting the velocity, amount of points completed, gets reflected on. After that point, three points get discussed. First of
all, what went well during the Sprint? The answers to that question should get repeated during the project. The second question is what could have gone better? The importance of this question lays in understanding why the answered things could have gone better. Elements can only be improved when the entire team has understanding of it. The last question is what the team will do differently the next sprint? There are always some points that can be improved. A few of these points will be picked by the chairman of that meeting to be done better immediately during the next Sprint. After each Sprint the team has gained more knowledge. Not only of the product, but also about the capabilities of the other team members, the velocity and the feedback gained. This process, from the start of the sprint until the end of it gets repeated every time. This makes the team stronger and the development will get improved by every sprint.

**Scrum Summary**

This section about Scrum concludes with a bullet list of all major scrum-settings that apply to our effectuated way of applying scrum in this project.

- The role of Product Owner has been assigned to Mourad.
- The role of Scrum Master has been assigned to Wesley.
- Tasks can be derived directly from the requirements, subrequirements or project meeting.
- The effort of completing a certain task is measured with points.
- Each point corresponds to 4 working hours.
- There will be a weekly sprint planning, preferably partially in the presence of our stakeholder. (They should not be part of the internal sprint planning, such as the division of tasks to a developer.)
- There will be a daily scrum meeting, for the development team only, to update one another and monitor each other's progress.

### 4.2.2. Trello

Trello is a project management tool used to define, order and distribute tasks [20]. A board corresponds to a project. Boards contain lists of cards, where cards are the tasks. Trello is made for the use of dividing and assigning tasks. However, for our project Scrum was enough to do such a thing. However, Trello was very useful to share information. It was used in such a manner that when one of the team members got stuck or gained additional information about a task or tool, he could create a card and put the information in it. This gave us the possibility to share information without interfering with each other. Especially during the first two weeks it was of a great help. Several programs needed to be installed and all installation links were posted at Trello. This way any team member, anywhere, could visit the Trello board and get the information he required. If it was not clear enough, then a team member could talk with another and ask him about it. It greatly improved the productivity of the team since there was little to no interference.

### 4.2.3. Revision Control with Git

We used Git for code revision control. The reason why we used Git is because all team members have had some experience in working with Git. On the plus-side of Git, we can all work in so called ‘branches’, of which the master-branch will always contain a working prototype. As discussed in the section on Scrum, we will increment this prototype with more functionality in each iteration. For every new task or a set of tasks, most of the time one or a subset of the (sub-)requirements, new branches will be created. In order to increment the master-branch with the additionally implemented functionality, we have set up the following sets of rules, defined in the following story:

1. The master-branch contains a stable version 0.X where X can be any positive number.
2. Developer Y wants to implement a new feature on the software product in the master-branch and branches from the master-branch to a new branch Z containing a logical name, where the name says something descriptive about the feature Y will be implementing.
3. Y will work alone or together with other developers from the development team on branch Z.
4. At the end of a sprint, developer Y has to pull the master-branch and merge the master-branch into branch Z and resolving all conflicts (if any).

5. Y has to checkout the master-branch and then merge branch Z into the master-branch, resolving all conflicts (if any).

6. With every merge, all unit tests need to pass on at least the master-branch.

7. Y updates the master-branch to version 0.(X + 1)

8. Y optionally deletes branch Z.

4.2.4. Code Style
When working in a group on large pieces of code, all code becomes more clear if the programming style of each team member is consistent with one another. For this to be realized we have set up some ground rules regarding code style and code conventions.

- **Comments**
  Each method and class should at least contain a JavaDoc-description describing at least the parameters a method consumes, a return type and which exception the method possibly could throw. Moreover a class should have a JavaDoc-description describing the function of the class, who uses the class and how that specific class is currently being used. Methods should contain inline-comments describing intra-unit functionality.

- **Naming**
  Each method and class should have a descriptive name in terms of it’s functionality or purpose. Class-names start with a capital letter, method names start with a lowercase letter. Names are built in Camel-Case. Test-classes should be named the same as the class they test, including the word Test. The same accounts for methods.

4.2.5. Product Timeline and Tasks
This section will show the initial timeline of our project. Table 4.1 states briefly our initial timeline where tasks are divided per week. List 4.2.5 will define each task, corresponding with a submodule in our DokterService. The sketch of the DokterService to which each sub-component belongs can be seen in figure

<table>
<thead>
<tr>
<th>Week</th>
<th>Iteration</th>
<th>Milestone/Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>N.A.</td>
<td>Project Demarcation</td>
</tr>
<tr>
<td>2</td>
<td>N.A.</td>
<td>Research</td>
</tr>
<tr>
<td>3</td>
<td>#1</td>
<td>Research + Program Outline + Basic Skeleton of Interacting Components</td>
</tr>
<tr>
<td>4</td>
<td>#2</td>
<td>MongoDB + Parser</td>
</tr>
<tr>
<td>5</td>
<td>#3</td>
<td>Web Interface 1 + Bro + Suricata</td>
</tr>
<tr>
<td>6</td>
<td>#4</td>
<td>Network Analysis + General Document</td>
</tr>
<tr>
<td>7</td>
<td>#5</td>
<td>Cut PCAP + Web Authentication</td>
</tr>
<tr>
<td>8</td>
<td>#6</td>
<td>Extended Analysis + Web Interface 2</td>
</tr>
<tr>
<td>9</td>
<td>#7</td>
<td>Oinkmaster + Web Interface 2</td>
</tr>
<tr>
<td>10</td>
<td>#8</td>
<td>Finishing up and preparing for the final presentation</td>
</tr>
</tbody>
</table>

Table 4.1: Initial Project Timeline.

Definition of Tasks
The following list will define the tasks as present in table 4.1. Each task belongs to a certain system component which can be found in figure 4.1.

- **Project Demarcation**
  Demarcate the project. Determine the extends of the project, such as size, importance and basic outline. Communicate this with the KPMG stakeholders and TUDelft coach. Also create a project timeline (table 4.1) which will be presented to both the stakeholders and the TUDelft coach.
• **Problem Description**  
Create a clear overview on the project. Create system diagrams.

• **Basic Skeleton of Interacting Components**  

• **MongoDB**  
Integrate MongoDB JavaDriver in our product. Be able to flush documents to a database.

• **Parser**  
Parse IDS output into an application independent format, which can be stored into the database and eventually be read in a report.

• **Web Interface 1**  
The application can be ran as a server, and IDS services can be initiated via a web browser. Web interface can also print (some) contents of the database.

• **Web Interface 2**  
Create a user friendly web interface, where a user can edit program settings, start IDS services and view reports.

• **Web Authentication**  
Authenticate both the web interface as server services properly.

• **Bro**  
Run the Bro IDS service and generate json-format outputs.

• **Suricata**  
Run the Suricata IDS service and generate json-format outputs.

• **General Document**  
Create general documents which contain the most useful information of Bro and Suricata, merged into one document based on timestamp.

• **Network Analysis**  
Integrate our own analysis into the DokterService.

• **Cut PCAP**  
Cut a .pcap-file according to the "interesting" results from Suricata, Bro and our own analysis. This should be done based on timestamp and Destination IP. For packet content analysis we will store the .pcap-contents in the database.

**INITIAL FULL SOFTWARE COMPONENT ARCHITECTURE**  
The following figure images our first and initial full sketch of the DokterService, which we used to create and divide the different tasks.
Figure 4.1: Software Component Architecture on Settings
This chapter proceeds on the previous chapter in such a way that it will reflect on the previously stated project strategy, with special attention to our software development methodology.

5.1. **Reflection on Software Development Strategy**
As stated in section 4.2 we used the scrum methodology as a project framework for our software development. Using scrum had really helped us by formalizing a project structure. By the end of the project we finished 8 successful iterations.

At the start of iteration one, we had sketched a rough backbone of our program on a very high level. All the components in the backbone could then be divided into subcomponents, which were divided among the four of us. This 'backbone' was being developed in this first week, in such a way that components could mutually communicate to one another. This was all going well. At the end of this week, we have met a new stakeholder and KPMG coach, which was absent during the first three weeks of our project. Meeting this new stakeholder changed some of the requirements and insights of the dokterservice. New features arose and other features seemed unnecessary to implement. After a meeting with all the stakeholders we concluded that we needed a new research-phase in which we analyzed a new perspective on the DokterService. The extra research phase resulted into the finalized requirements as they are stated in 2.3 and resulted in the product sketch as can be seen in figure 3.2. The research we conclude from this phase is also contained in chapter 3. After finishing the new outline, which will be presented in table G.1 in appendix G we were already halfway week 4 of the project and had almost finished the week what was supposed to be iteration #2.

At the start of iteration #3 we wanted to fully adapt the scrum methodology, in order to achieve the best and utmost results during our development process. We were motivated to work even harder on the project because we had encountered a two week delay, due to received changes on the requirements and the DokterService in general. Applying the scrum methodology happened as planned and discussed in section 4.2.1.

At the beginning of an iteration, during the weekly sprint planning meetings, tasks were created together with the entire team. In the same meeting the tasks were created, tasks were also estimated and assigned to an owner. Tasks could also be assigned to multiple owners, where the estimation of effort accounts to a single task, instead to the cumulative estimation of each task owner. Figure 5.1 shows a snippet of our product backlog in iteration 8. The most important thing to notice in this figure is that for each task, a status will be maintained. The status will be maintained by the owner or owners of that specific task, in order to communicate to one another who is working on which task. The Product Owner also checks the states of the tasks from time to time to ensure everything is going alright.

5.2. **Demo**
Apart from the weekly scrum meetings, we also organized several demos with our customer. During those demos the customer will not only be hearing about the progression of our process, but the customer will also be interactively 'playing' with our product, discovering it's actual capabilities and it's limits. The whole development team will be present during this demo in order to explain certain features or implementations to the customer, receive feedback from the customer on the implementations and elaborate in a mutual discussion on any possible changes that should be adapted. As it turns out, during these demo's some of
the requirements were changed a lot. This will be discussed in section 5.3. During the demo’s even more parties, apart from our official stakeholders (e.g. our assigned KPMG coaches), were present as well. They were mainly interested in the development of our product, because they are the parties who will be utilizing our product. It was therefore very interesting for us to receive some feedback from them as well.

Eventually we organized three demos. Because our real software implementation started in week 5, we first of all had to create a demonstrable version of the DokterService. At the beginning of week 7, we showcased our first product to a collection of stakeholders and possible end-users. The stakeholders were amazed how rapidly we had implemented a workable prototype, which already ran the IDS services and included a web-based user interface. After the first demo, new requirements were discussed and some existing requirements were altered. The requirements that have changed will be discussed in section 5.3. The final requirements are stated in appendix F. The last two demos were held at the end of week 8 and also concluded in a change or addition of requirements.

5.3. CHANGE OF THE REQUIREMENTS

As mentioned above, the requirements were often changed. Changes of the requirements result in a higher customer satisfaction, as the product meets more specific the customer’s needs and wishes. A change of requirement is a pristine reason to apply the scrum methodology, as development tasks are flexibly generated. Hence during our development process we could easily apply such requirement changes. New requirements were always discussed and classified together with the whole development team in terms of feasibility. Depending on the result of classification, new requirements get an implementation status, which states the urgency of implementation. This section states the new requirements.

The following requirements have been changed or added. A full list of all final requirements has been included in appendix F.

1. The analysis report should contain:

- Top X lists, where X can be set through the user interface by the user.
- Top X lists, where a user can determine through the user interface if the list should be ordered ascending or descending.
- Top X lists of at least:
  - Most visited countries
  - Most visited IPs
  - Most used protocols
  - Most used ports
  - Most origination IPs
  - Most visited URLs
5. Process Reflection

- Largest files downloaded
- Most used User Agents (HTTP(S) traffic)

- Through the user interface, a user should include or exclude other fields.
- List of malware that was found (based on Suricata output; e.g. Critical / High alerts), which should be distinguish based on Suricata-output-type. The user must be able to filter these results.
- Visualize actual packet data with a certain time margin from the .pcap-file with CapMe. (Optional)

Instead of requirement 14

- The analysis report should contain:
  - Top 10 lists of:
    (a) Most visited countries
    (b) Most visited IPs
    (c) Most used protocols
    (d) Most used ports
    (e) Most origination IPs
    (f) Most visited URLs
    (g) Largest files downloaded
    (h) Most used User Agents (HTTP(S) traffic)
  - List of malware that was found (based on Suricata output; e.g. Critical / High alerts)
  - Analysis of bandwidth usage (optional).

2. Output formats of the report should contain:

- Table contents in the reports should be downloadable in a CSV-format.
- Generated charts should be downloadable in a PNG format.
- A world map colored according to the amount of source and destination IP’s originating from the corresponding country. (Optional)

Instead of requirement 15. The output format of the report is PDF in a pre-defined template. Preferably also a format which is editable (.doc(x) or otherwise editable is fine)

3. The micro instance is required to get mounted to a specific static directory. This is user instantiated by terminal commands. Instead of requirement 3. The captured data is transferred from the micro instance to the analysis instance by means of an automated system that is user initiated
6

PRODUCT

6.1. User Interface
This section provides an overview of the user interface of our developed software product. Using the user interface a user is able to perform several actions, utilizing the application. The most important part of the user interface is the fact that you can run the services and generate the reports. The UI consists of several webpages, each with its own functionality and role. The creation and structure of these pages can be found in section 6.2. The upcoming sections discuss what these pages show and how to navigate through them.

6.1.1. Login Screen
When starting the application, this is the first page you see. You are asked to fill in credentials. Upon providing correct credentials, further access into the system is provided. If this is not the case, every other page will result into a '404 not found' error. Showing 404 errors makes sure an invalid user is not able to know which URLs are valid and which are not. Credentials are stored in a specific database. More information about this database can be found in section 6.4. Input credentials are matched with the data in the database. If there is a match, the user will be marked as logged in using a cookie containing a unique SessionID. This procedure is further explained in section 6.2. If this cookie is acquired, you find yourself in a valid session. The login screen disappears and a new page is created. This page contains several options discussed below.

6.1.2. Run Services
The first action discussed in 'Run Services'. As the name states this action runs the services. When starting this action, it finds the .pcap-files and transfers these to a local directory, as discussed in section 6.3. After the transfer the services are ran on these .pcap-files. Section 6.3 also explains what the procedure is to go from running the services and having the data stored in the database. After the data is stored in the database, this action is done. Throughout the process, a log keeps the user up-to-date about what is currently happening in the system. It shows for example which thread is currently running, when it is done, when the system is parsing the data and when the data is exporting the data to the database. A 'refresh logs' button refreshes the showed log panel, to show the most up to date log information.

6.1.3. Generate Reports
When the database is filled with data, reports can be generated. There are three different reports available: General Report, Malware Report and Analysis Report. Section 6.5 explains thoroughly how these reports are generated and what information resides in these reports. Every report has the same format. It has a settings panel on the top of the report containing all adjustable settings. Under this panel, the information is shown in table format. Each table corresponds to a field. Each row contains an element and information about this element is divided in the different columns.

The 'General Report' contains TopX information about the several fields stored in the database, based on the occurrences of unique elements in those fields. The settings for this report consists of the fields shown, how many elements should appear per table and if the elements should be sorted in ascending or descending order. The default settings are 8 fields on, 10 elements per table and in descending order. To avoid having an enormous report, the amount of elements per table can be set to a low value. Based on the information
searched for, a user now also has the ability to sort the elements to his liking. The actual data is represented in tables. Each field has its own table containing a ‘name’ and ‘count’. The elements are sorted by count. These tables can be exported to a csv format using the ‘Export as CSV’ button. This makes it possible for the user to acquire the data in a editable format.

Lastly there is a ‘Generate Chart’ button above every table. This then opens a new window containing a pie chart of the data in the table. Each element in the pie chart corresponds to a element in the table and the value of each element corresponds to the count value. This pie chart can be edited. Clicking on an element disables it in the pie chart, which can lead to a clearer pie chart, if for example a big element is disabled. This pie chart can also be downloaded, using the ‘save’ button in the window. This downloads the pie chart in PNG format.

The second report is the ‘Malicious Report’. This report contains the malware information stored in the database. Because there are several malicious categories to check on, you can select which category you want to show. Each category has its own settings. For example, the category ‘DNS’ has the options to view DNS queries and answers and show different fields and the category ‘TLS’ has no adjustable settings. Some options also contains a filter. ‘Fileinfo’ allows to filter by extension and size of the file to narrow the results shown. Some fields have extra information per element. When clicking on such an element, it expands and the extra info is shown. For example when showing DNS info, by clicking on an element you are able to see which queries and answers were requested. As for General Report, this report also has a ‘generate csv’ button per table, which exports the table into csv format, which can be used for editing. This report has an extra feature. When clicking on a timestamp, a new window opens. This window shows the pcap file containing the timestamp and the information stored in the database corresponding to this timestamp. This is a initial start for requirement 14.

Lastly there is the ‘Analysis Report’. This contains information about the analysis done by the application itself. This includes for example a list of Source IP’s with the least/most DNS requests and a list of least/most used User Agents. The only adjustable setting for this report is the setting to sort the information in ascending or descending order. Some elements also have extra information. This can be shown in the same way as in ‘Malicious Report’. For example, in table ‘Source IP’s with amount of DNS requests’, clicking on a element shows the DNS Hostnames which where queried to. Again each table can be exported to csv format. Section 6.4 explains that some analysis fields can be filtered using a whitelist. On the top right part of this page a button is available, which direct to a new page. This page contains an expandable panel for each whitelist. When expanded, this panel shows all values in the whitelist, which can be checked and deleted using the ‘Remove’ button from whitelist. The whitelist can also be expanded, by filling in a value and clicking the button ‘add’. The list is then updated with the new value.

6.2. **WEB ROUTING**

As been stated before in section 3.3.4, from requirement 8 stated in section 2.3 can be derived that our stakeholders want to access and control our developed software product via a web-browser. For this to be possible, we had to create a ‘local server’ which is accessible by using several commands on some local address. The local server is for obvious reasons ran on localhost. Our server listens to requests on port 8080 on localhost. While this port is easily changeable, we have used port 8080 as this port is a popular alternative to port 80 for offering web services and it is above the restricted well known service port range (ranged from 1-1023) [21]. This section will further elaborate on the existing ‘*Rest-classes and their corresponding URL and purpose. The asterisk in ‘*Rest’ represents a wildcard for any prefix, meaning all classes which are an instance of the super-class Rest. Lastly section 6.2.1 will describe our implemented way of the construction of new webpages.
**REST instances**

Figure C.3 in appendix C gives an architectural system component overview on the system packages required for web-routing services. When a user navigates with a browser to a valid URL, the user invokes the corresponding Rest-instantiation, requiring some action or computation. Note that all Rest-instantiations are a child of Rest and therefore are a kind of Rest. Table 6.1 lists all of the Rest-classes with their corresponding URL. If the user goes for example to http://localhost:8080/ the user invokes the IndexRest-instantiation, which in turn determines what to do and what to show in the browser.

<table>
<thead>
<tr>
<th>Class</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>IndexRest</td>
<td><a href="http://localhost:8080/">http://localhost:8080/</a></td>
</tr>
<tr>
<td>ProgramRest</td>
<td><a href="http://localhost:8080/program/">http://localhost:8080/program/</a></td>
</tr>
<tr>
<td>SettingRest</td>
<td><a href="http://localhost:8080/settings/">http://localhost:8080/settings/</a></td>
</tr>
<tr>
<td>LoggerRest</td>
<td><a href="http://localhost:8080/roller/">http://localhost:8080/roller/</a></td>
</tr>
<tr>
<td>AuthenticateRest</td>
<td><a href="http://localhost:8080/auth/">http://localhost:8080/auth/</a></td>
</tr>
<tr>
<td><strong>Report specific</strong></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.1: List of *Rest-classes and their associated URL's

Except for the Index-instance (for logging in) and the AuthenticateRest-instance, all other *Rest-instances require a logged-in user. Since valid session is shared information to every Rest-instance, on every possible request (POST, GET) fired by a user, the system will invoke their super-class information by validating the user's browser's cookie with SessionManager. Depending on the cookie-validation and request-related information, a WebPage instance will be build from template HTML-, CSS- and JavaScript-files, which will then be shown in the user's browser.

**Using cookies**

As stated above, except for logging into the system, all *Rest-instances require a valid session. As told in section 6.3, valid user sessions will be stored in a browser cookie. Authentication will be dispatched by the AuthenticateRest class, which creates a new session in SessionManager if and only if the user can be validated. Cookie content validation or session validation can be done in any *Rest class, because each *Rest class extends the superclass Rest, which interacts with SessionManager. SessionManager can then return to the Rest-instance whether a user has validly logged in or not, which leads to what page to be shown. If the user credentials can be resolved, a new session will be created and maintained by SessionManager, which will be valid by default for the duration of:

- 30 minutes,
- until the user closes the browser,
- until the user logs out.

Valid sessions will have an ID which will be stored in a cookie in the user's browser. From now on, every time a user performs an request on a valid subdirectory on http://localhost:8080/, each acting *Rest class will invoke the SessionManager by checking the session-ID from the cookie contents. From now on every description of the components will assume the user is properly authenticated. AuthenticateRest will then redirect the user to the IndexRest page with a valid session stored in a cookie. Every invocation of a *Rest-instance will then validate the cookie as has been explained above.

Since validation of logged in users will be done based on SessionID's, one could also see the possibility of brute-forcing these SessionID's until a valid and active one will be generated. This is actually a desperate case, because of the possible amounts of SessionID's. SessionManager generates a 32 character long with a variety of 32 characters. Hence there are $32^{32} = 2^{160} \approx 1.5$ quindecillion (49 non-decimal digits) possibilities. Figure 6.1 shows an example of a SessionID cookie, with a 32 characters long SessionID.
WEB NAVIGATION

However multiple services of our program are accessible through direct and full HTTP URLs, the services are all made to be available from the index page via the user interface. Figure 6.2 shows the interconnected relationships from Rest-instances. It makes more sense to talk about interconnected Rest-instances since we do not want users to enter the URL's immediate from a static context and moreover the Rest-instances will be the dynamic engine behind a page. Hence the figure entities are Rest-instances to demarcate a Rest-triggered service, rather than full URL's as the latter says nothing to the end-users.

From figure 6.2 one can immediately see that once entering localhost:8080, one automatically triggers IndexRest which will show a login form if the user has not logged in yet. By submitting the login form, one will invoke AuthenticateRest, which will subsequently edit the user's browser's cookie with a valid session. After the cookie is successfully set, the user will be redirected to the IndexRest-instance, but now the user will be shown a full service-featered webpage. The user can now navigate to other *Rest-instances contained in the red dashed container in figure 6.2. If a user tries to access an authorized-only page for which authentication is required, the user will be prompted a 404 File Not Found page. The reason why a 404 will be shown, instead of a 403 Forbidden (in case of an illegal request) or an 401 Not Authorized (as normally the case
for legal requests by unauthorized users), is to deceive possible directory traversal attackers by deceiving them no possible content is present.

6.2.1. Sitebuilding
As stated in the previous section, each *Rest class will be the dynamic engine for creating a webpage. To construct our webpages we have made some HTML, CSS and Javascript templates which will be used throughout our program. Their content will be loaded, edited and returned, resulting in a full webpage. CSS and Javascript are added for a better user experience, and in order to gain a clear overview on the webpage. Template HTML documents contain tags, which will be replaced with content by the Rest-instance.

Example of sitebuilding
To clarify how the webpages are build, we will discuss one example in which tags are replaced in order to dynamically create a webpage. This section will focus on one part in the template.html shown in figure 6.3.

```html
<!-- Succes and Eror message section of template.html -->
<div id="succesMessage" class="message">
  ~succesMessage~
</div>
<div id="errorMessage" class="message">
  ~errorMessage~
</div>
```

Let’s say the code content of figure 6.3 is loaded by an instance of SettingRest. Next analyze the use case where an end-user tries to create a new DokterService account (handled by SettingRest). If all required fields are correctly filled out and the form to create a new user is submitted, the end-user will be redirected to the setting-page, but now with a new succesMessage stating that the new user account has successfully been added to the database. SettingRest, which also initiates the user account addition, knows the status of the addition, and replaces the "succesMessage" tag with a succesMessage. Note that tags will be identified by a single wave character on both ends. If no errors occurred, no error messages will be set, meaning the errorMessage-tag will be deleted. Any Rest-instance deletes by default all their applicable tags, which are unset or not implemented at all. Hence no error message will appear at all, however a success message will be present.

6.3. IDS Services
This section will describe the cause of actions which are a result of starting the services, as discussed in section 6.1.2. We call this section the IDS Services section, because it will describe how the adapted Intrusion Detection Systems, will be initiated, managed and executed. A graphical system component overview to illustrate this section can be seen in figure C.1 in appendix C.

6.3.1. Directories
Because many processes will output even more files, the directory structure needs to be clearly stated and documented. This subsection describes the reading directories used for the input of captured network traffic and the working directories for each IDS service instance.

Reading Directory
Prior to the execution of the IDS services, the .pcap-files to be analyzed to be analyzed need to be contained in a certain directory. The .pcap-reading directory is for any system executing DokterService /docterservice/pcaps/. Before starting the IDS services, it is then required to transfer the captured network traffic to the .pcap-reading directory. In a production environment where the Micro-Instance contains the captured network traffic, the Micro-Instance should first of all be mounted to that specific .pcap-reading directory. The files should be transferred via a bash script or mount command.


**Working Directory**

The subsections subsequent to this section describe how all IDS services will be initialized. Every mentioned service will output its logs into a log-directory located in `/docterservice/logs/`. To view any IDS generated output, one has to look for the log-directory.

### 6.3.2. Managers

When the user has properly authenticated himself, the IndexRest class will respond and prompt the user with the option to start the services. If the user chooses to start the IDS services, ProgramRest will respond to this request and initiate a SuperManager. SuperManager in turn initiates a Manager. This Manager works in different states, namely Initialize, Update, Work and Finished. The states will be explained below.

1. **Initialize**: A manager will be clearing the folder structures and indexing the .pcap-files. This indexing is done by the usage of a tool called "capinfos". This tool gets ran on every .pcap-file first and the start-time -and end-time, together with the name of the .pcap-file to the database. This happens really fast and does cost remarkable computational time. This information, now stored in the database, can be used later in the program. More about the usage of this data can be found in section 6.5.4.

2. **Update**: A manager will perform all one-time updates. Currently implemented one-time updates are the updates on the sets of rules from `emergingthreats.com`, which need to be updated prior to the start of all IDSs.

3. **Work**: A manager will instantiate multiple NetworkToolManagers, each NetworkToolManager will be assigned a set of .pcap-files to process.

4. **Finished**: A manager is finished, all created NetworkToolManagers are also finished.

Manager updates SuperManager on each change of state. The Update state of the Manager initiates all tool-specific update-actions. As discussed in section 3.3.1 Suricata needs to run against the most up-to-date set of rules. This update needs to be done once, prior to the analysis, hence the update is done by Manager. The Work state will initiate a concurring amount of NetworkToolManagers for each predefined amount of .pcap-files. If for example, 220 .pcap-files need to be processed and every NetworkToolManager is assigned 11 .pcap-files, 20 NetworkToolManagers will concurrently execute all network tools on their set of assigned .pcap-files.

A NetworkToolManager is instantiated by a Manager and is assigned a set of X files. The NetworkToolManager has three states: Initialize, Work or Export. In the Initialize-state, the pcaps are indexed. In the Work-state, a manager will initialize, manage and run all of the adapted NetworkTools for each single .pcap-file from their assigned set of .pcap-files. A NetworkToolManager will listen to all managed NetworkTools, in order to know when all NetworkTools are finished processing the .pcap-files. Lastly the Export-state flushes the hashmap accumulated with merged network information.

### 6.3.3. NetworkTools

Currently implemented/adapted IDS services are Bro and Suricata. Both have their own classes and the both inherit from the abstract superclass NetworkTool. Because of the latter, Bro and Suricata also have different states in common, namely Work, Parse, Export and Finished. Each of these states will be explained below.

1. **Work**: A NetworkTool instance will execute the corresponding IDS services associated with the services. I.e. Suricata will search for matches against a set of malicious rules.

2. **Parse**: A NetworkTool will parse their produced output with a mapper corresponding to the instance of NetworkTool to a general object. Parsing will be done with the help of tool-specific object maps, which will be further discussed in section 6.3.4.

3. **Export**: A NetworkTool will flush and merge all general objects to the database.

### 6.3.4. Parsing Output

All network tools produce certain output by generating one or multiple log-files containing the output data. Because we have edited some of the running configurations, these log-files will be decomposed of JSON formatted entries. These JSON-entries will then be read by each corresponding instance of NetworkTool line by line.
line, where they each are parsed to a general object, consisting only of the fields of relevant data. Because the output from Bro is in the first instance independent of the output from Suricata, the keys format in both distinct log-files will be different as well. The same might account as well for some values. The latter is the case at the different timestamp formats Bro and Suricata wield in their log-files. Bro formats its timestamps with in UNIX format, whereas Suricata formats its timestamps in a ISO 8601 format. We have chosen to parse all timestamps to UNIX timestamp if they are not in that specific format. The reason for this is that Bro logs will always be larger than the logs produced by Suricata, because Bro monitors and outputs all network traffic, whereas Suricata only outputs (potentially) malicious network traffic and alerts. Because Bro's logs will by default be always larger than the output of Suricata, the only way to optimally decrease performance consumption, is to convert Suricata's timestamp to Bro's timestamp. The same applies to all kind of keys and values produced in any log-file. After this process, all parsed documents can be exported to the database.

6.3.5. **Exporting Parse Results**

Each networktool then has a set of parsed and generic objects, which now have to be stored and merged into the database. Objects that describe the same event will be merged, as they will be the sum of all key-value pairs from both objects. The merging of documents and flushing data to the database is handled within the database component as described in section 6.4.1. Network tools will simply pass on a list of general objects.

6.3.6. **Observer Pattern**

As becomes clear from the aforementioned provided information, running the IDS services will result in a chain of instance creations, where some of the instances need to observe the instances it will create and other instances need to notify their creator. Table 6.2 quickly summarizes the instances that will be created, and the interconnected observe and notification connection.

<table>
<thead>
<tr>
<th>Instance</th>
<th>States</th>
<th>Notifies</th>
<th>Observes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SuperManager</td>
<td>N.A.</td>
<td>Manager</td>
<td>N.A.</td>
</tr>
<tr>
<td>Manager</td>
<td>Initialize, Update, Work, Finished</td>
<td>SuperManager</td>
<td>NetworkToolManager</td>
</tr>
<tr>
<td>NetworkToolManager</td>
<td>Work, Finished</td>
<td>Manager</td>
<td>NetworkTool(s)</td>
</tr>
<tr>
<td>NetworkTool</td>
<td>Work, Parse, Export, Finished</td>
<td>NetworkToolManager</td>
<td>N.A.</td>
</tr>
</tbody>
</table>

Table 6.2: Summary of Interconnected Observe Instances

In order to achieve a clear overview, many components need to observe other components. Therefore we have adapted the Observer Pattern [22]. The Observer Pattern needs two interfaces, one subject and one observer. Realizations of a subject notify realizations of an observer if the observer had initially added himself as listener. In our implementation, Runnable is the interface which must be implemented by every subject realization. Implementing Runnable for subject realizations is killing two birds with one stone, because we also want exactly those components to run multi-threaded. The interface for observer realizations will be ThreadListener. If any class implements ThreadListener it can listen to all Runnable-instances it has access to. Figure 6.4 clearly illustrates this scenario. SuperManager implements ThreadListener, creates a Manager which is a Runnable and gets therefore notified by Manager on every change of state. Manager also implements ThreadListener, which enables it to listen to NetworkToolManager, which is a realization of Runnable. The chain goes than on and on, until we have reached a NetworkTool instance, like Bro, which is because of dependency a realization of NetworkTool, hence also a realization of Runnable.

The reason why there has been implemented a hierarchical chain of observers, is because Manager can create unlimited NetworkToolManagers and NetworkToolManagers can create unlimited NetworkTools. In order to achieve some regulation, many instances can be created. They need to update their instance creator on any change of state, whereafter their creator will decide what to do next. However currently the amount of NetworkToolManagers is limited in the amount of .pcap-files to be processed and the amount of NetworkTools is limited in the amount of implemented networktools, the amounts can get uncontrollably and tremendously high.

6.4. **Database**

In order to store data and information, the application uses several databases. As stated in section 3.1, MongoDB is used in our application. MongoDB uses a hierarchical structure, with Databases, Collections and Documents [5]. The structure of our database system and each individual database will be explained. We
also created a Query class in order to communicate with the databases. This will be explained in the last section.

**6.4. DATABASE STRUCTURE**

The structure contains three different databases: 'DataDB', 'UserDB' and 'AnalysisDB'. The purpose of these databases can be extracted from the name, but will further be discussed in the upcoming sections. For each database, a specific class is created to communicate with the database. An overview of this structure can be seen in figure C.5. The first item to discuss is the interface 'IDatabase'. This class contains all methods a database class should have. This interface is very useful for testing. Next is a 'Database' class which implements IDatabase. This class contains a 'MongoClient' attribute, which is implemented using the Singleton pattern. This because there can be only one MongoClient. One could ask why this class is not abstract, since it only functions as a parent. This is because at the start of the application, a MongoClient has to be started. Since this has to be database independent, this is done by the Server class using a Database object. The 'DataDB' class, 'UserDB' class, 'AnalysisDB' class each correspond to one of the databases and extends the Database class.

**DATABASE DATA**

As the name states, this database stores the data. In Section 6.3.4 is discussed how the data is created and parsed. After the data is parsed, it is exported to this database. This database has one collection, which then contains all data exported by the different IDS services. This data is used for creating reports, performing automated analysis and performing manual analysis. The data is stored until an authenticated user clears the database. Because of the procedure discussed in section 6.3, this data is stored in a generic format, which is tool independent. Data from different services which correspond to the same event are merged. This reduces the amount of conflicts which can occur. The database class that communicates with this database is the 'DataDB' class.

**DATABASE USER**

This database stores all user credentials, which can be used to authenticate a user. These user credentials are stored within one collection in the database. This is also the only collection in this database. When logging in, which is described in section 6.1, the credentials provided are compared to the credentials in this database. If a match exists, the user is authenticated and can proceed using the features of the application. Section 6.1 also explains that an authenticated user can add a new user and remove existing users. Performing these actions will update this database. The user database does not contain actual passwords, but contains hashes of passwords and matching salts. When providing a username and password, a document will be searched for that contains the username. If this document is found, the hash value is calculated using the stored salt.
and provided password. The database class that communicates with this database is the 'UserDB' class. This class uses the NewUser class to insert new users in the database. The NewUser class contains methods to calculate salts and hashes, which can then be inserted in the database.

**Analysis Database**

The Analysis database is used to store information used in the analysis of the data stored in DataDB. Currently it contains only whitelists, used during the analysis discussed in section 6.6. Each whitelist corresponds to a collection in this database. This collection contains a name and a value, which is then checked for during the analysis phase. Collections can be added to this database to contain extra information such as new whitelists or other conventions. The database class that communicates with this database is the ‘AnalysisDB’ class.

**6.4.2. Query**

In order to query the DataDB, the class ‘Query’ was made. This class is able to create an aggregation function which can be used to retrieve specific data from the database. This makes it much easier to create aggregation functions, since creating these manually can sometimes be very complex. It also contains predefined queries that are most commonly used, which can lead to a much faster and easier construction of queries. In order to generate the data that is used in the different reports, the query class is extensively used. For example, when generating most occurrences of certain fields in the general report.

**6.5. Report**

For the DokterService, the gathered and processed data should be visualized and some meaning has to be created for the crunched data. This is done by the creation of different types of reports. These reports are divided in: the General Report, Malicious Report and the Analysis Report. This distinction in three makes it possible to focus the attention to specific parts of the information. The distinction is made by looking at the information that has to be displayed. In subsection 6.5.1 the creation of so-called documents is explained. Documents are, in terms of the reports in DokterService, classes which perform the initialization of required data for the report. Next, the different types of reports are going to be elaborated on in subsection 6.5.2. Within the page for the Analysis Report, it contains the ability to maintain a whitelist for specific information. This whitelist and the purpose of it are elaborated on in subsection 6.5.3. Finally, for the Malicious Report a feature has been implemented to find in which of the .pcap-files, which are processed, a particular event has been triggered. More about this feature is to be found in 6.5.4.

**6.5.1. Document Creation**

Before any report is generated, on the background an initialization of required data is prepared. This is done by the class called ReportDocument. This class is a superclass three types of documents that need to be created. The documents are: GeneralDocument, MaliciousDocument and AnalysisDocument. The superclass handles most of the processing. However, it might be the case that in the future a new document has to be prepared for a report which requires specific initialization. That is where it is useful that the children of the superclass can override or extend the functionality of the pre-defined methods. For the general report, the HTML-output is less specific than for the malicious report. So, moving up some methods that are reused by other classes is where the strength lays of the ReportDocument superclass.

The ReportDocument does a few initializations:

1. The retrieval of the fields set at the top of the page. (More about this can be found in 6.7.3).
   Before the class performs any action on the data stored in the database, it collects information set at the top of the reports-page. Every reports-page has its own settings that can be set and thus this varies per report. However, there is a class, namely PostData, which can process the settings for every report in ease. It splits the data and stores it in a HashMap, which has key value-pairs. After the PostData-class has finished processing the settings, it is passed on to the functionality that communicates with the database.

2. Storing elements which contain all the fields required to set up the table content.
   Based on the pre-processed setting data, it might be the case that some fields are not required to be shown. The ReportDocument then filters all the fields away which are not required. By means of this filter, the total search through the database also gets a speedup. When all the filters are applied and only the required fields remain, these elements get stored in a List and ready to be used.
3. **Production of HTML-output per element.**
   
   At the final phase, all elements are collected in a List and they are iterated over. For every element it should be possible to create the HTML. This HTML-generation is done by the usage of a class called Table.java. This class provides the ability to create a table with a title, headers and rows containing the required information. This takes the load off from the ReportDocument-class and puts it in hands of the Table-class. This results in a decrease in unit sizes and an increase in readability and maintainability.

**MALICIOUS DOCUMENT**

In the case of the MaliciousDocument, queries need to be set up and therefore, they have another layer between the report generation and the document data gathering. This is defined as an Event, the class is also named Event. The creation of this additional layer provides an improvement of maintainability, the code structure is clearer and the unit sizes decreases. This Event-class is an extension of the ReportDocument-class. There are six events: *weird names*, *alerts*, *DNS*, *TLS*, *file information* and *HTTP*. Therefore, every event has its own class which extends from the Event-class. This specialized class contains specific queries to gain the required data. In a specific event-class, like for example Alert.java, it is required to fill in the method to gather the data and name the event. The table generation is all done in Event.java. This is done in this manner to avoid duplication of code in every child of the Event-class. Every Event produces the HTML in the same manner. The HTML is then accessible for the Malicious Report which gets the HTML in the same way the General Report does that.

**ANALYSIS DOCUMENT**

Analysis Document is slightly different from the two previously discussed Documents. There is no iterative process of gathering the data as in the previous Documents. However, here it is required to add elements hard-coded. The analysis that is done, discussed in 6.6, is done on very specific types. These types should get added in their own way. However, when the elements are added to the Document, the HTML is generated in the exact same way as is done for the Analysis Document. This makes this class the easiest editable and maintainable of all the Documents which are created.

**6.5.2. DIFFERENT TYPES OF REPORTS**

There are three different types of reports: the General Report, Malicious Report and the Analysis Report. In the following parts these three reports are being discussed separately.

**GENERAL REPORT**

In the General Report a top-10 list, gathered and crunched by the GeneralDocument, is displayed. All elements for the top-10 list are iterated over and their HTML is collected. The collection of their HTML is eventually viewed to the user. The General Report contains, like every other report, an own individual .html-file. This file contains this report specific information. This is primarily done by using tags within the .html-file. These tags then get processed within the report at java-level and replaced with information that belongs at the place of the tag. When this all is done, the total of HTML is shown to the user. For more information about the view of the General Report, this is discussed in 6.1.

**MALICIOUS REPORT**

The Malicious Report provides the user with information regarding the events that are triggered by the tools. Especially the events triggered by Suricata are of value to the user. Those events need more attention and therefore set in this report. There are six different types of events that can be toggled on/off. The events are: *weird names*, *alerts*, *DNS*, *TLS*, *file information* and *HTTP*. In case one of these events is toggled on, a table of the event-specific data is shown. As for the General Report, the data that is required to view to the user is being processed by a ReportDocument-class. In this particular case, the MaliciousDocument-class. This class performs queries, specific to the chosen event, and fires this query to the database. The database responds to the query and gives a result. For every piece of result the MaliciousDocument also knows how to generate HTML for it. The subclasses of the MaliciousDocument, if required, can override this functionality in case the superclass does not produce the correct HTML output. This provides the developers the ability to generate output as specific as required. As for the previous report, the Malicious Report also has its own .html-file. This file also contains report-specific tags and information. When all the tags are processed and replaced, the eventual output is shown to the user.
**Analysis Report**

The Analysis Report is the report that contains any suspicious activity in the .pcap-files which are gathered based on analysis done by our team. The suspicious activity that has been found is presented in, like the other reports, a table-format. In this case, the data that is required to be viewed is processed and delivered by the Analysis-class.

**6.5.3. Whitelist**

At the page for the Analysis Report, a button is added to manage the whitelist. The whitelist is a list, for multiple fields, containing field values which are considered harmless. For example, HTTP User Agents sometimes are malicious. However, with this whitelist, which is maintainable from the interface, those in the whitelist are not shown for the Analysis Report. This results in only showing the fields which are not found in the whitelist-database. The same can be done for hostnames and many different fields depending on the implementation of the Analysis-class.

**6.5.4. .PCAP Information**

For the Malicious Report a unique feature was added. DokterService runs on a large amount of .pcap-files. These .pcap-files do not have a specific range of time, but rather get filled until they reach a certain limit of packets/bytes. This caused an issue, when a certain timestamp should get matched with one of the .pcap-files. However, as discussed in 6.3, the .pcap-files are being indexed at the beginning of the system. At the Malicious Report, when the user decides to investigate in which .pcap-file a certain event has occurred, it is possible to click on the timestamp. When he clicks on the timestamp a new, small window is opened which gives the .pcap-file in which the timestamp has occurred. This makes it possible for the user to then search, with another 3rd party tool, within the .pcap-file and look deeper into the traffic of that packet. This reduces the amount of time that is required to find the event and makes the overall analysis much faster.

**6.6. Network Analysis**

The application is able to perform analysis based on the information gathered by the several services, such as Bro and Suricata. An overview of how Network Analysis takes place in the application can be found in figure C.4. When authenticated, a user can generate three different kinds of reports based in the information in de database, as stated in section 6.5.2. One of these reports is the ‘Analysis Report’ which contains information about the analysis done by the application. The procedure to generate this document contains several actions such as interacting with the database, generating the report and actually performing the analysis. These actions will be explained in this section.

**6.6.1. Analysis**

Prior to generating the document, the data stored in the database has to be analyzed. This is done by the Analysis class. This class contains static analyzing methods, which each analyses the data to show potential threats. In order to acquire this information, the Analysis class makes use of a DataDB instance. The DataDB instance is able to retrieve the data from the database, so the Analysis class is able to perform its analysis. This is a list of the analysis performed:

1. **Suspicious User Agents**
   - Different people use different user agents. But in general people tend to use a user agent which is commonly used. This analysis checks how many times a user agent is used and shows suspicious user agents based on that fact. The result is filtered with user agents by choice

2. **Suspicious Source IP’s and amount of DNS Requests**
   - Sometimes people with malicious intents try to send data and information through DNS requests. This type of analysis checks how many times a specific source has used DNS requests. The amount of these DNS requests can be suspicious.

3. **Suspicious Files**
   - Files can have different sizes, but some files are suspicious if they have a certain extension and a lower or higher than a corresponding predefined average size. This type of analysis checks for certain extensions and their corresponding suspicious sizes.

An extended list of the analysis we wanted to perform with explanation can be found in section 3.4
6.6.2. **Information Gathering**
The Analysis class communicates with two Database classes. First there is the DataDB. This is the gateway to the actual data created by the services. This data is used for analysis. As explained in section 6.4, to communicate with the database, the Query class is used. The Analysis class creates a query object and adds queries to this object using the corresponding methods. The query can then be executed on a DataDB instance. This way the data needed can be retrieved from the database. Secondly there is the AnalysisDB. For certain analysis extra information is needed. For example when searching for least used User Agents, there could be the need to filter valid User Agents. Such a ’whitelist’ is stored in the AnalysisDB. More info about this whitelist can be found in section 6.5.3. When creating the query, extra matches are added to the query, so the elements in such a whitelist are filtered out of the result. So for retrieving the data which is analyzed, the DataDB class is used and for retrieving the whitelists used to filter the results, the AnalysisDB class is used.

6.6.3. **Generating the Document**
Now the application is able to perform the analysis using the database, but this information also has to be visualized. This is where the AnalysisDocument class comes in. This class extends from the ReportDocument class, since this class contains the methods to generate documents. Section 6.5.1 explains how the data created is visualized in the Analysis Document.

6.7. **Settings**
In this section we will elaborate on all the different settings that can be chosen from in our application and how it is handled. A quick overview of the settings flow can be found in figure C.2.

6.7.1. **SettingProcessor**
On the settings-page the user can set and edit many system configurations. Because the system is user authenticated (as described in section 6.2) logic dictates that other user accounts should be able to be managed as well. Therefore a user must be able to add or remove existing users from the user database. He can also add more trusted http-user-agents or URLs from DNS-requests to a whitelist in the database. Sometimes the user also wants to do a quick analysis over a large quantity and size of .pcaps. This is difficult to achieve with a lot of IDS-tools running. So in the settings-page the user can also enable and disable certain tools to do the analysis. This can all be done using the SettingProcessor class as seen in figure C.2 and is the only other internal class (except for *Rest-classes) the user has interaction with. To make these actions the SettingProcessor needs to have access to the SystemConfig class.

6.7.2. **System Configurations**
Tools like Suricata need to be updated prior to its execution (requirement 11), but this is not always needed, e.g. when the user wants to rerun the services right after it has been ran. Updating Suricata also consumes a lot of time and this is nice to be avoided when doing a quick analysis. So the user can also uncheck Suricata to not update before its execution using the SettingProcessor. But for this the SettingProcessor needs access to the SystemConfig which keeps track of this information via ToolInfo which in turn keeps a list of all tools (ToolList). The user can also specify which fields the user deems necessary to be shown in a report. This will be explained further in 6.7.3. So SystemConfig keeps track of all this information via FieldInfo which in turns keeps a list of all fields (FieldList).

When modifying the settings, SystemConfig uses Setting to pass on changes made to the lower components, e.g. when checking Suricata to update prior to its execution SettingProcessor sends SET_UPDATE down to the flow to Tool, which sets this option to active so when the services runs and checks on tools that need to be updated Suricata gets returned. The same can be said for when a user checks the boxes to show certain fields in the general report, but this time it uses SET_ACTIVE. In the General Report the user can also choose how many items the top x (x is the number of chosen items) lists must contain and this is done using the SET_TOPX Setting.

There are 3 types of reports the user can generate. When viewing these reports the user can choose as mentioned before which fields he wants to be shown in the report. Not all fields are necessary to be shown in a specific report (mentioned in 6.7.3). But the user can still choose which fields should be showed in the report. FieldInfo keeps track of which fields are constrained to which report via Field. This is the reason why Field is subdivided into three subclasses, namely MaliciousField, AnalysisField and GeneralField. To which report they belong can be derived from their names.
6.7.3. REPORT CONFIGURATIONS
As mentioned before, there are three different reports. Each report has different settings to be chosen from. In the following subsections we will further elaborate on the different setting configurations each report has.

GENERAL REPORT
General Report is a report containing lists of the most or least used values. Some fields are not always necessary for a client, so the user must be able to choose which fields are necessary to be shown in the report. He can do this by clicking on the settings option above the report and checking or unchecking boxes next to the field names present in the settings section. When the user hovers over these field names a small box appears with a description of said field. Besides these actions one thing to be noted is that not all fields that are stored in the database are necessary to be shown in the General Report, e.g. it is not feasible to show a top 10 list of the least or most used timestamps, so this field is not present in the settings section. All this information is stored in FieldInfo and is handled by their respected Rest-classes which uses SystemConfig to attain this information.

MALWARE REPORT
For Malware report the settings section is different. First, the user has to choose what kind of event (Weird, Alerts, DNS, TLS, Fileinfo, HTTP) should be viewed. Each type of event has different settings. When clicking on 'Show results' after choosing a setting a POST request is sent and processed by ReportsRest which returns the desired result.
When choosing the Weird or TLS report there are no extra set of settings, but Alerts has one extra setting. The alert event are produced by Suricata's analysis. Every alert has its own severity level and with the extra setting the Alert report provides the user is able to decide what severity is important to be shown.
In DNS report the user can choose what type of DNS events should be shown. He can choose between Query, Answer or Both. The user can also choose what fields are important, namely rrname, rrtype and rdata. (What these fields mean can be found in Appendix D.3.)
In Fileinfo the user gets prompted with a query in which the user can specify what kind of files should be shown and also a constraint considering the size of the file (this can be used for their own analysis e.g. a PNG file that is bigger than 20 MB sounds suspicious).
Lastly the HTTP report also has its own extra set of settings. The user can choose which fields are important, namely User Agent, Length and Status. The results can also be filtered based on the HTTP status code. The user can also choose to see a specific status code or exclude it from the results.

ANALYSIS REPORT
In Analysis Report the user can choose the see the results in ascending or descending order. This is helpful for e.g. source IP's that send a high amount of DNS Requests or a low amount. Next to this setting the user can also manage the Database's Whitelists 6.5.3. The user can view/add or remove values from this Whitelist and filter the results based on the Whitelist.

6.8. EXTENSIBILITY
One of the requirements our stakeholders retained towards the software we wrote, is the notion of modular building. The main reason why our code should be build in a modular way, is that the stakeholders can maintain the product in such a way that the IDS services can easily be extended. In this section becomes apparent that our code has been build in a modular way and is therefore easily extendable with other IDS services. Section 6.8.1 discusses how current IDS services have been build in a modular way and also explains how to extend the running IDS services in the future. Section 6.8.2 elaborates on the extensibility for parsing the output, as this will also be necessary to maintain general documents in the database.

6.8.1. IDS SERVICE EXTENSIBILITY
As illustrated in figure C.1 and discussed in section 6.3.3, each IDS service is an instance of a NetworkTool. IDS services are build modular, meaning that if another IDS service needs to be added to the DokterService. One just needs to extend NetworkTool and implement some IDS dependable functions. All instances of NetworkTool will then be dynamically loaded by a NetworkToolManager, meaning that extra IDS classes which are an extension of NetworkTool will also be loaded automatically.
In order to create an extension of NetworkTool, one has to implement all abstract methods from NetworkTool. We call the abstract methods from NetworkTool "IDS dependable functions". The IDS dependable
functions are also stated in figure 6.5. These must be implemented for each instance of NetworkTool, in order for the DokterService to know how to run the extra IDS services. These functions provide therefore running- and parsing configurations. The method getRunCommand will tell DokterService the running command of that specific IDS service. Method getExtension provides DokterService of the extension of the log-files an IDS service may produce. For example, Bro outputs its files in a log-extension, whereas Suricata creates an output contained in a json-file. Method parseDocument parses IDS specific fields to generic documents. The importance of and reason for parsing has also been discussed in section 6.3.4.

```java
public class NetworkTool{

    ...

    //Returns the running command of the IDS service
    public abstract String getRunCommand(executionPath : String);

    //Returns the extension of the log-files
    public abstract String getExtension();

    //Parses the parameterized document to a general document
    public abstract void parseDocument(document : Document);
}
```

Figure 6.5: Abstract Methods from NetworkTool: The IDS Dependable Functions

### 6.8.2. Parsing Extensibility

Parsing is done with a mapper corresponding to the IDS service output. For the IDS service Bro there has been created a BroMapper. The same accounts for Suricata. An IDS specific mapper simply contains a HashMap, which maps fieldnames from the output to general fieldnames. These will be maintained in the database. For example Bro’s output contains a field called ts describing the timestamp of certain events. Suricata’s output contains a field called timestamp also describing the timestamp of certain events. In order for these to match in the database, the ts-field is mapped to timestamp-field by the Bro-mapper.
This chapter will provide information regarding the validation of each requirement stated in appendix F in appendix F. Section 7.1 will contain a list of validations per requirement, if possible. Section 7.2 provides some future work recommendations as well as new insights for any probably additional future extensions of our final software product.

7.1. Requirement Validation
Every section will first state the final requirement, then the section explains how the requirement has been validated.

1. Data input consists of multiple .pcap-files that contains captured network traffic.
   If a user runs the services via the user interface, the application searches in a predefined reading-directory (/dokterservice/pcaps/). It finds every single .pcap file that is stored inside the directory and runs the IDS services on these .pcap-files. This way the data input can include multiple .pcap-files. This is described in section 6.3.

7.1.1. Requirements

2. .pcap-files are parsed with the original timestamps.
   During the parsing phase, the log-files which are generated by Suricata and Bro are being parsed to a generic format. This is then exported to the database. During this phase the timestamps are also converted to a generic format (UNIX time). This corresponds to the original timestamps. The process of parsing fields, and more specific, timestamps is discussed in section 6.4.1.

3. The micro instance is required to get mounted to a specific static directory. This is user instantiated by terminal commands.
   Section 6.3.1 describes that when running the services, the application searches for a specific read-directory (/dokterservice/pcaps/), which should contain all .pcap-files to be processed. Therefore the micro instance has to be mounted to the read-directory.

4. The information stored in a database should be an application independent format.
   During the parsing phase of each NetworkTool instance all log-files are parsed to a predefined generic format by use of a mapper. This makes the data application independent. Section 6.3.4 elaborates on how application-dependend data is parsed to a general document by utilization of IDS-specific mappers.

5. The database should scale well for large amounts of data (> 10 GB).
   The application makes use of MongoDB. MongoDB is very well known for its scalability and good processing capabilities of large amounts of data (see Section 3.1.1).

6. The application should be built in a modular way, allowing addition and removal of modules. (e.g. different modules for the different functionality, extra NIDSs).
The application is build in such a way that every service is ran by a NetworkTool object. This object is started by the Manager. If the need exists to add a new service, a new class should be made, which extends the NetworkTool class. This way the Manager can run the service. The way these objects are being created is described in section 6.3.2. Removing a service is just a matter of removing the class itself. If the service should be turned off and not removed entirely, the only thing to do is to configure the manager accordingly (This can already be done using the web interface). Section 6.8 discusses how our application could be extended.

7. Computation time should not exceed 2 days (48 hours).

After performing some benchmark tests on the development server, we can verify that the processing of 1 TB of .pcap-files takes less than 48 hours. The performance can even be boosted if some IDS configurations are tweaked even further. Moreover the actual production server where our product will be ran, the hardware specifications are even better. For instance the development server consists of 4GB of RAM memory, whereas the real production server is contain of 32GB of RAM memory. We therefore are able to verify that the computation time does not exceed more than 2 days.

8. The application is web-based. No command line interaction should be necessary to upload a .pcap-file, generate a report or view any retrieved data.

The application is using the RESTful style. The front-end of this application is a fully web-based interface. This interface has a couple of options. First there is `run services`, which runs the services on all .pcap files which are in the specified directory. Secondly there is `retrieve report`. This leads you to a page where three different reports can be generated. In these reports fields can be set on and off, which means all data stored in the database can be viewed in these three reports. Further information can be found in section 6.1.

9. The access to the web-based application is properly authenticated.

Section 6.1 explains that when starting the application, it checks if you are correctly logged-in using a cookie. If not, you have to specify the right credentials to log in. If this is done, the browser stores a cookie with a SessionID. Everytime a service of the application is called, it checks if there exists a cookie with a valid SessionID. If this is not the case, the application throws a 404 error. All valid credentials are stored in a User Database using MongoDB. More information about the cookie management can be found in section 6.2.

10. Malware analysis is done with at least:

- a Bro
- b Suricata

The application runs the services by making the Manager create NetworkTool instances. This is explained in section 6.3.2. The subclasses of the NetworkTool instances are then used to specify which service has to be run. There are two child classes of NetworkTool. Firstly there is a child which updates the rules used by Suricata and is able to run the Suricata service. Secondly there is a child which is able to run the Bro service.

11. The rule list is updated prior to the start of an analysis.

Suricata matches network traffic against a set of rules describing malicious network traffic. The most up to date rules will be downloaded from http://www.emergingthreats.net/. When starting the Suricata service, the application first updates the rules by retrieving the rules currently used by emergingthreats 3.3.2. Because this can take up some time, an option has been implemented to skip this when running the services.

12. Automatic file carving (i.e potential malicious files are carved from the traffic and stored on disc) (Optional)

Since this was an optional requirement, this requirement was not prioritized. Due to lack of time, we were not able to implement this feature.

13. Automatic virus scanning of carved files (Optional). Since this is an extension of the previous requirement, implementing this feature was not feasible.
14. The analysis report should contain:

15. Top X lists, where X can be set via the user interface by the user.

16. Top X lists, where a user can determine via the user interface if the list should be ordered ascending or descending.

17. Top X lists of at least:
   a. Top 10 lists of:
      i. Most visited countries
      ii. Most visited IPs
      iii. Most used protocols
      iv. Most used ports
      v. Most origination IPs
      vi. Most visited URLs
      vii. Largest files downloaded
      viii. Most used User Agents (HTTP(S) traffic)
   a. Via the user interface, a user should include or exclude other fields.
   b. List of malware that was found (based on Suricata output; e.g. Critical / High alerts)
   c. Visualize actual packet data with a certain time margin from the .pcap-file with CapMe. (Optional)

The application is able to generate three different reports. First there is a ‘General Report’ 6.5.2. In this report a top ten of specified fields is generated. This consists of all fields which are stored in the database and includes points i to vi specified in the requirement. The last two points are not included in the General Report, but can be seen in the other two reports. Secondly there is a ‘Malicious Report’ 6.5.2. This report is based on the information found by Suricata. The data is formatted in a way the stakeholders were satisfied with. This includes Alerts, HTTP information and other information specified by Suricata. Lastly there is an ‘Analysis Report’ 6.5.2. This report shows information based on our own analysis using the data provided by Bro and Suricata.

Each report has settings implemented in the page. Section 6.7.3 explains which settings are available. This section also validates the requested settings requirements.

The last requirement can be validated partially. We were not succesful in integrating CapMe in our product. But we were able to create a initial start in order to finish this requirement which is explained in section 6.1.3. Requirement 14 was optionally and has been added after the last demo we gave. Since the new requirement was reciever on such short notice, we were not able to perform a research on the program CapMe and it’s functionality. Therefore we agreed with our stakeholder that CapMe-functionality should not necessarily be implemented into the DokterService. It was sufficient to show the information in which .pcap-file a corresponding event belongs. If deemed essential, this feature could then be implemented in the future.

18. Output formats of the report should contain:
   - Table contents in the reports should be downloadable in a CSV-format.
   - Generated charts should be downloadable in a PNG format.
   - A world map colored according to the amount of source and destination IPs originating from the corresponding country. (Optional)

Section 6.5 explains how the reports are created and section 6.1.3 explains how these reports are visualized and what are the possible report configurations on said reports. This also explains that tables in each report are exportable to a csv file. It also states that each table can be viewed as a pie chart, which can be downloaded in a PNG format if needed. Due to lack of time, the world map requirement has not been validated, as has been agreed upon with our stakeholders.
7.1.2. CONCLUSION
After analyzing the validation on all final requirements we can conclude that we have implemented all the feasible and non-optional requirements. Some of the optional requirements were not met because they would require additional research. This has been well communicated with our stakeholders, whereas most optional requirements were added in the nick of time.

7.2. FUTURE WORK RECOMMENDATIONS
At the end of this project, we were able to implement almost all requirements. But this does not mean our product is fully complete. There are a couple of extra features which could be implemented to make this product more user-friendly and functional.

One of these features is an integration with CapMe [23]. CapMe is able to open a .pcap file on a valid timestamp, which could be useful to a user. Each element in a report contains a timestamp. The stakeholders requested a feature where clicking on a timestamp would open the corresponding pcap, showing the event containing this timestamp. This could be implemented using CapMe, but due to the lack of time, this was not feasible. But in the current version, clicking on a timestamp opens a new window which shows the pcap the event occurred in and the document stored in the database containing this timestamp. One could extend this existing feature, by integrating CapMe in it.

Another feature that was considered is generating a world map containing data about where source or destination IPs originates from. Continents or countries would be colored according to the amount of originating IPs. This way one could detect strange destination or source countries via a visualized interactive global map.

The last addition we will discuss is extension on the Analysis Report. Compared to the analysis items that has been discussed in section 3.4, the Analysis Report (section 6.6.1) is always extendable. Implementing the full list or even extending the existing list would be our final future work recommendation.
This chapter will explain how we as a software development team assured the utmost quality of the developed code. To ensure this quality we did some research on the topic of “quality” in the context of software. From that research we concluded that quality can be defined by 5 standards set up by CISQ, the Consortium For IT Software Quality. In a paper released in 2012 they defined these standards as: Reliability, Performance, Efficiency, Security, and Maintainability [24]. Based on each of these standards, we integrated a system in our development process to ensure that each standard is assured. According to CISQ, they justify these standard as:

“The purpose of these measurement specifications is to create standards for measuring Software Quality Characteristics that are automated, objective, economical to use, and technically feasible.”

One popular way to assure code quality is by integrating tests alongside the development of software. We applied several different kinds of testing, all of which will be further elaborated on in section 8.1. The next approach to assure the standards are met, is to actually analyze the quality by means of software metrics. These metrics can be calculated by a free-to-use plugin provided by Google to Eclipse, called CodePro Analytix 8.2. These metrics are first of all calculated by the development team. Altough, they also are measured by the Software Improvement Group (SIG) 8.2.1. Next, the development team has set up rules in the context of integrating new components to the system. These rules provided reliability to both the stakeholder as the team in terms of stability and always having a working version. These rules and the approach of integrating the components is will be discussed in section 8.3.

8.1. TESTING
At the early stage of the development tests were taken into account. Testing the product is an essential part of the development of the product. It gives insight in both the adaptability and the unit size of the methods being tested. The intention of testing is to find software bugs. All humans make mistakes, and some of those mistakes are unimportant. However, it might be the case that some of those mistakes are of great importance to the product. They can be expensive and even dangerous. Where expensive means taking a lot of effort to repair them and dangerous means the whole system crashing or failing due to the mistakes/bugs. Testing provides both the developers and the customer reliability and satisfaction in the product. Moreover, it might be the case that some software parts, during the development, were not checked thoroughly. These parts might not fail on a developer’s local system, but might on the one of the customer. This happens since the local system of the developer is an ”perfect environment” for the software to be run on. The reason why developers most of the time have the ”perfect environment” is because developers can tweak all local settings. However, with testing it is possible to create an environment that is not perfect. Even though the environment is not perfect, the product should be executed in an expected manner.

Furthermore, to gain more confidence on the software quality that is created we used Continuous Integration (CI). With the use of CI failures and errors in the code are detected early and located more easily. However, when using CI it is of importance that the development team holds to several rules and their responsibilities.
Continuous Integration (CI) is a development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early.

The automatic test running facility from the CI will also be utilized for performing regression testing. Running automated regression tests is one of the seven principles of software testing defined by Bertrand Meyer in his article Seven Principles of Software Testing [25]. The goal of regression testing is to test if all unmodified code is still working. Since a CI server will run all unit tests by default, they could also function as regression tests.

In this section several aspects of testing will be discussed. First of all, how our team integrated the process of testing into the development of the product. Secondly, what types of tests were performed and why these types were chosen. And In the end will be discussed which testing frameworks have been used for testing.

8.1.1. INTEGRATION OF TESTING

Every member of the software development team had gained experience in writing software tests prior to this project. This means that testing should not be of major difficulty to be integrated in the software development process during this project as well. During the first few weeks, the software was not thoroughly tested. Testing was done by two people of the team at the end of a sprint. This is of course not the proper way to integrate testing in the flow of a software project. The use of testing at the end of a sprint also brought some complications with it. At the end of a sprint the testers discovered flaws and improper parts of the code. This led several times to refactoring and changing multiple big chunks of the code. So, this was a bad way of testing. Not only was it hard to test parts of the system, but it also resulted in actual addition of the developed code.

In week 3 a CI server was introduced and integrated into our system. This would lead to finding these flaws and errors more frequently and faster than just at the end of the week. This was already a big change in the fashion in which code was made. Parts of code, and tests, that were written were based on the “perfect environment” earlier mentioned. For example, converting timestamps was done based on the local system. However, on a CI-server the time zone was different making chunks of code return unexpected timestamps and these parts were also failing on the CI-server. This triggered the event for the entire team to carefully write code and also dig more into the documentation of several methods.

A few weeks later the team decided to change the frequency of test-writing more. Too few tests were written and the coverage gained by this was also terrible. A discussion was held with the team and new rules and responsibilities to each member were set up. One of the most important rules was that a new/edited component of the system will not be integrated with the stable version until thoroughly tested. This enforces the team to think even more about the structure and design made.

8.1.2. TYPES OF TESTS

During the development of tests, the team uses both static and dynamic testing. Static testing is done by means of pair programming. A pair of developers go through the code and do a review and walkthrough about the logic of the code. Not only the logic is being checked, but they also perform a verification of the code. Dynamic testing is done by writing test cases for each method on its own. This is done by component testing, also known as unit testing. Specific functions of a method are tested. The methods are tested in such a way that the desired output is always gained. Both in the case of failing and passing.

Software testing methods are traditionally divided into white- and black-box testing. These two approaches are used to describe the point of view that a test engineer takes when designing test cases. With white-box testing the point of view is the internal structure of the system. However, with black-box testing the developer has no knowledge of the internal structure and is only aware of what the software is supposed to do and not how it does that. During the development of the product the focus was set on white-box testing. In case the internal structure of the system fails, it also has an effect on the “black-box” environment.

8.1.3. TEST FRAMEWORKS

UNIT TESTS AND MOCKING

As discussed earlier, the focus was set on white-box testing. To perform this as good as possible, the used testing framework was JUnit 4. Together with this unit-testing framework, PowerMock was used. JUnit is primarily used to perform unit tests and see which unit tests were failing, giving errors or passing. The entire team has worked with JUnit before and it has also been recommended from the Delft University of Technology.
8.2. CODE PRO ANALYTI X

In the Bachelor course named Software Quality and Testing to use it. However, some methods in the system were hard to test or not possible to do only with unit tests. To fully test the functionality of those methods it was required to mock objects. An object subjected to a test might have one or multiple dependencies. With the use of mocking these dependencies are taken away and “imitated”. By simulating the behavior of the real objects, the tests are easier to create and more specific cases can be tested. To exemplify the latter, if we want to test the parse-behavior of a NetworkTool-instance, we probably do not want to write actual test-data to the database. The database-instance is then being mocked, in order to gain database-instance-behavior, but not actual data flushes to an existing database. In order to be able to use mocks, we used Mockito, a mocking framework for Unit Tests in Java [26].

**CODE COVERAGE**

Integrating code coverage alongside unit tests, is a good habit in order to gain even more confidence in the code. We used EclEmma [27] to analyze the code coverage of our product. Analyzing code coverage during the development process can have multiple advantages. The biggest reason for us is to make sure every (new) component is (considered) tested. Code coverage gives a clear overview of which components are tested thoroughly and which components are not tested at all. The latter is the case when new code has been written without any tests, or that specific component is impossible or too difficult to test. A code coverage report clearly maps these issues.

Another reason to adapt code coverage in combination with unit tests, is to pinpoint functionality and make sure functionality never changes. When for example a part of the written code should be refactored, unit-functionality or unit-behavior should not change. If unit tests have been written to test such behavior, they should in general still pass after a refactor or modification on the code. A code coverage report can then function as a monitoring system that the modified units are indeed tested. If we happen to notice in a code coverage report that tests are routinely not being written or omitted, we can take proactive action, such as setting up revised testing ground rules, pair-programming testing, etc. This way of adapting code coverage reports had also been introduced by Andrew Glover as Evaluating Code Quality with Code Coverage Reports [28].

8.2. CODE PRO ANALYTIX

For the DokterService development the tool that is used, is called CodePro Anal ytiX [29]. CodePro AnalytiX is currently in the hands of Google. With the use of the tool, which is easy integrated into Eclipse, it was possible to view the code quality of the product. The tool is extensive in the features it provides. It has seven key features, which all provide a part to get the most accurate measurement for the five standards. The most used features within CodePro AnalytiX are the ones to measure metrics, similar code analysis and the code audit feature. The first one, to measure metrics, consists of small automated tools that measure and report on key quality indicators in the produced Java source code. Figure 8.1 depicts an example of output generated by the Metrics feature of CodePro. These metrics provide information based on the metric set inside CodePro. This can vary from the number of methods inside a class to the cyclomatic code complexity within it. All the metric results together add up to the performance, reliability and the maintainability of the code. The second feature that is used most often is the Similar Code feature. This provides the possibility to make CodePro go through all classes and find similarities. It happens fairly often in software projects that code is copied and pasted. This decreases the maintainability of the product. Instead of a change in one place, it is required a change in multiple regions. Changing multiple classes for one reason is of course a bad way of code design. The last big feature is the one to perform Code Audit. This feature shows warnings, with three kinds of severities, if the implemented audit rules are violated. This is very useful in cases that parts of the product are written but not carefully looked thought out. An example is in cases that variables have a 1-letter name. This makes it very hard for later improvement to figure out what that typical variable has of meaning inside that scope.

8.2.1. SIG

**FIRST SUBMISSION**

Before the 6th week of our project we had to upload our code to Software Improvement Group (SIG) whom evaluated our code and gave us the rating 4 out of 5 stars based on their own analysis. From Appendix A can be derived that the code we uploaded was above average in terms of maintainability. The highest score (5 out of 5) was not achieved because of two points, namely: Duplication of code and large Unit Sizes. After this first
evaluation we have prioritized in fixing these issues and maintaining them during our development phase. Next, both points will be discussed in how they apply to the developed code and the approach we have used to fix them.

**Duplication of code**

Duplication of code is one of the most occurring issue in software development. This is mainly because it is much easier to copy and paste code where required than to create a structure in the code to avoid this problem. Throughout the development we tackled this issue several times However, it seemed that we did not address it enough. The moment we received the results from SIG we re-evaluated the code and we found several code duplications throughout the code. Most of them were purely based on the structure of the system that was done poorly. One of, and the biggest, places that had duplication was found in a HTML-class. This class was intended to take of the load to create HTML-code by hand every time it was required. However, the class itself had multiple code duplications to create this HTML-code and the classes that used the HTML-class most of the time created the same HTML. This was bad practice since the software was hard to maintain in this manner. The second, most occurring place, for duplication was in the classes that handle the tools. Those classes contained a lot of exact same code. The proper way to handle this is to move code that is similar to their superclasses and let them be called.

To tackle this re-occurrence of duplication of code, we organized a meeting with all the developers of the team and reviewed the structure of the system. After a few hours of putting everything on paper and map the dependencies, we agreed on a solution. The classes that required to generate some HTML got their own superclasses. This way the children of that class extended from the superclass and set their variables right. The method in the superclass then are called. This gave us both more flexibility and assured a better maintainability. In the future, the code only needs to get adjusted at one single place, instead of multiple places. The same solution was applied to the tool-classes. After integrating this solution, the classes became much smaller and also provided readability. Thus, this concluded the research to where and why code duplication occurs and the solution we came up with.

**Unit Size**

Unit size applies to two different types of measurements. The first one is applied to the size of packages constructed in the software and the second to the size of classes. Starting off with the first one, it is bad practice to have unbalanced package sizes. This, most of the time, indicates that a component of the system performs much more than other components. It is, for example, not a recommended way to have a server perform a lot of computations and communication, while another component which could take the load off from the server is performing under average. This unbalanced structure can be traced back also in the comparison between classes in a package. Having a class that almost acts like a God-class, while the others are just simple POJO-classes (Plain Old Java Object) breaks the balance in the structure. Both measurements of unit sizes effect the maintainability and the reliability of the system. A big package or just enormous classes are hard to maintain. People intend to leave them as they are and actually add more code to them, since this is much easier than splitting them up. It is unstructured to find where a call is made. Aside from the
maintainability it also effects the reliability of the system. With a huge pressure on a specific component of the system, it is also more likely to break, contain bugs or even crash. This makes the system unreliable and both the developers and the stakeholders always want a system they can rely on. In our project the package containing the REST-classes, which handled all the GET- and POST-requests with the web-interface, was too big in unit sizes according to the results from SIG. We also agreed with their conclusion. This, like with the point of duplication of code, meant that we did not carefully structured our code to prevent this from happening. However, it came to the point that it occurred. During the meeting our team had to discuss the duplication of code, we also discussed unit sizes. Instead of looking into the classes which were having too big unit sizes, we went through our system and listed what each component should do. After that was noted, we went more into detail and looked into what each class should do. When this was done, we looked into the troubled packages and classes with too big unit sizes. It immediately became clear where some methods should get moved to. Some classes were also added to the system to create more structure. At the end this resulted in having a much more balanced, maintainable and reliable structure.

SECOND SUBMISSION
Two weeks after the first submission, we were again required to send our code to the Software Improvement Group for a second and final evaluation. They again gave us the rating 4 out of 5 stars. They also stated that the code volume has doubled, while the overall maintenance has stayed the same. The main problems in our code was again: Duplication of code and large Unit Sizes. We fixed the items that caused a decrease in code quality from the last evaluation, but at the same time we created new large units and duplicates. Therefore the score stayed the same.

Duplication of Code
The adjustments and improvements that were integrated into our development process were not sufficient enough. According to the results, returned to us by SIG Appendix B, other occurrences of duplication of code occurred. Most duplications from the last submission were removed. However, this was likely to happen. The amount of code that was developed doubled in size within two weeks. These two weeks contained several additions of features and with this duplication of code re-occurred. SIG mentioned four specific examples of classes which contained duplication of code. These examples are: Bro.java, Suricata.java, AbstractThreadTool.java and AbstractThread UserManager.java. For two of these examples, namely AbstractThreadTool.java and AbstractThreadManager.java, a day after submission to SIG got refactored. This refactor got focussed on moving methods higher up the hierarchy. It appeared to us that several methods were available in other classes in the same extent of content. This was a wake-up call for us to refactor these particular classes but also review other classes which are set up in this type of manner. The amount of duplication of code has been decreased dramatically since the first submission.

Unit Size
Furthermore, according to the second evaluation by SIG, the same occurred for Unit Sizes. The classes containing large unit sizes which were mentioned in the first submission got handled. However, new large unit sizes had been introduced into our system. The methods of large unit sizes were mostly the ones which contained switch-cases. These methods were required to switch between very specific cases. Besides these types of methods, also methods within the Rest-classes were of great unit sizes. To battle the issue of unit size, the same methodology was used as discussed for the first submission. We have seen that the unit sizes decreased with this methodology. But it seemed to not be significant enough to wipe the issue of unit sizes off the map. An additional approach we introduced was to critically look how easy or fast a new feature could be added to the existing system. If this seemed to be hard, it might be the case that the unit sizes are too large and should get split up. These two measures together should be enough for the system to tackle the issue of unit size.

8.3. INTEGRATION
In the previous sections several approaches have been provided to assure quality of the product. To optimize this quality the entire team agreed upon four rules to keep the quality maximized. They have been set up in discussion with all team members and only integrated when all members agreed on them. Next, the four rules are going to be summed up.

1. **Structure diagram required for the new component.**
   In addition to a new component or feature being assigned to a team member, the structure should be drawn out before the start of code development. This forces the member assigned to the task to carefully think out the structure to prevent any issues in the future. When the structure is set up, it
should also be discussed with another team member. This creates four eyes to look into depth of the component. This minimalizes any chance of code being constructed in a poor manner.

2. **Reusable or extendable.**

Any newly made component or piece of code should be either reusable or it should be easily extendable. This minimalizes the chance of duplication of code. Also, since it is reusable, it only needs editing at one particular place in the system. Thus, it adds up to the maintainability of the system.

3. **New components require to be tested thoroughly.**

For every new component that is being developed, it requires to be fully tested. By testing the component thoroughly, it assures that the components behavior is as expected. Any exceptions or errors that it might throw, should be handled correctly. An untested component does not get the right to be integrated with the most up-to-date and stable version of the Dokterservice. The tests and the new component should also not interfere with previously written tests. Their behavior should remain the same. This in total provides reliability to both the team and the stakeholders.

4. **Audit and metric analysis.**

At the end of the development of the component, after the three previously mentioned rules have been applied, a final rule has to be applied. The tool CodePro AnalytiX, discussed in section 8.2, should be run against the system. The most important features that need to be applied are the ones for getting audit information and metric analysis. The results gained after the analysis must be compared to the results before the implementation. The differences, if any, should be explainable.

In case that all the four rules are followed up and used, the new component/feature is allowed to get integrated with the stable version of the system. When the integration is performed, the team members should pull from that new stable version and work further on it. This assures everyone of working on the new, most up-to-date and stable version.
The DokterService project at KPMG has been a challenging experience to us. The goal of this project was to design and develop a "Security Information and Event Management System" (SIEMS) that manages Network Intrusion Detection services and provides a report specifying the condition of the analyzed network, hence project DokterService arose. In order to develop such a SIEMS, a list of requirements are created together with our stakeholders at KPMG. In the end we managed to meet all listed feasible requirements and deployed version 1.0 of the DokterService on our stakeholder's server. We soon expect version 1.0 to be extended, because that is the way DokterService has been developed. Our client wanted special attention on the ability of extensibility of the product, to integrate additional IDS services. Because we have build the DokterService in such a modular way, extension of the DokterService can be easily achieved.

In the beginning of the project we explored the fields of network analysis and experienced several new tools and frameworks. We not only executed in-depth research on two well known IDSs Bro and Suricata, but also interviewed field specialists working at KPMG. One of the main advantages we profited from by working at KPMG was having a whole unit of experts available to us in order to answer any questions we had regarding their field of expertise. On the other hand, KPMG was happy to have us around, allowing them to ask questions to us regarding the DokterService project. As it turned out, many other KPMG employees did not even know about the DokterService project and were amazed four bachelor students were able to carry out such a project.

KPMG had already build and deployed a proof of concept of the DokterService, however the proof of concept consisted of a incoherent set of tools. Executing the juvenile DokterService was hard, because the system had to be managed by terminal commands. Adding additional IDS services was also a laborious act, as each service demanded specific configurations. Hence the DokterService should have been reorganized and rebuild from scratch with special notion regarding the extensibility and end-user experience of the DokterService.

As a project development methodology we have applied scrum to maximize the working efficiency of the development and frequently involve the stakeholders of the project. Requirements can change during the development lifecycle, as they should in order for the product to meet the stakeholder's expectations. This has happened often during the development of the DokterService, but because we have applied such an agile methodology, the development team could adapt to these evolvements in requirements. The scrum methodology therefore proved it's competence. The development team encountered some major evolvements regarding project implementation. One of them which enormously affected the process of our project, there was even a second research phase needed. But after performing the extra research phase and re-structuring our project's approach, we increased the efficiency of our process by applying clearer, smaller and more specific (sub)-requirements which were cleared and approved from all parties. Hence we can state conclude that applying scrum has helped the increase of efficiency and adaptation of evolving requirements during the software development project.

In the near future we expect the DokterService to be used extensively by KPMG. Even more because the product is easily adaptable and carriageably extendable to be modified for their own running services. Due to the easy to use web interface, complex manageable and configurable IDS services can now be executed with a push on the button.
The following paragraphs contain the feedback we have received by SIG on the first submission of our program's code. This feedback has been received by us on May 29th 2015 and is send by Dennis Bijlsma.

A.1. DUTCH

De code van het systeem scoort 4 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code bovengemiddeld onderhoudbaar is. De hoogste score is niet behaald door een lagere scores voor Duplication en Unit Size.

Voor Duplicatie wordt er gekeken naar het percentage van de code welke redundant is, oftewel de code die meerdere keren in het systeem voorkomt en in principe verwijderd zou kunnen worden. Vanuit het oogpunt van onderhoudbaarheid is het wenselijk om een laag percentage redundantie te hebben omdat aanpassingen aan deze stukken code doorgaans op meerdere plaatsen moet gebeuren. In dit systeem zit het grootste duplicaat in het genereren van HTML, in het bestand DocumentRest.java. Je kunt dit soort gedupliceerde codeblokken makkelijk naar een private methode verplaatsen, en dan die methode aanroepen. Dit voorkomt dat je toekomstige aanpassingen aan de gedupliceerde code dubbel hoeft te doen.

Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt. Binnen de langere methodes in dit systeem, zoals bijvoorbeeld IndexRest.generateHTML, zijn aparte stukken functionaliteit te vinden welke ge-refactored kunnen worden naar aparte methodes. Commentaarregels zoals bijvoorbeeld 'The element itself' zijn een goede indicatie dat er een autonoom stuk functionaliteit te ontdekken is. Het is aan te raden kritisch te kijken naar de langere methodes binnen dit systeem en deze waar mogelijk op te splitsen.

Tot slot is het goed om te zien dat jullie unit test-code hebben geschreven. Hopelijk zal het volume van de test-code ook groeien op het moment dat er nieuwe functionaliteit toegevoegd wordt.

Over het algemeen scoort de code bovengemiddeld, hopelijk lukt het om dit niveau te behouden tijdens de rest van de ontwikkel fase.

The message above is the original one received from Dennis Bijlsma. For convenience the message has been translated to English at the next page.
A.2. English

The code of the system scored 4 starts on our maintainability-model, which means that the code is above average maintained. The highest score has not been reached because of low scores for Duplication and Unit Size.

For Duplication the system looks at the percentage of the code which is redundant, in other words the code that occurs multiple times in the system and could be removed in principle. From the perspective for the maintainability it is desirable that the percentage of redundancy is low. This is because adjustments to these pieces of code require to be applied at multiple places. In this system the majority of duplication occurs in the generation of HTML, in the file DocumentRest.java. You could easily move these chunks of code to a private method, and then call that method. This prevents that you need to apply future changes to the duplicated code multiple times.

For Unit Size the system looks at the percentage of code that is above average long. Splitting these types of methods into smaller pieces results in the fact that each part is easier to understand, test and to maintain. Within the longer methods in this system, like for instance IndexRest.generateHTML, there are different pieces of functionality which could get refactored to different methods. Comment-rules as for instance "The element itself" are a good indication that there is an autonomous piece of functionality to be discovered. It is recommended to look critically to the longer methods within this system and to split those up where possible.

Finally, it is good to see that you have written unit test-code. Hopefully the volume of the test-code will also grow at the moment that new functionality is being added.

In general the code scores above average, hopefully you manage to keep up with this level during the rest of the development.
The following paragraphs contains the feedback we have recieved by SIG on the second submission of our program's code. This feedback has been recieved by us on.

B.1. Dutch
In de tweede upload zien we dat het codevolume is verdubbeld, terwijl de score voor onderhoudbaarheid ongeveer gelijk is gebleven. Jullie zitten dus nog steeds op 4 sterren.

Op het niveau van de deelscores zien we dat jullie bij Duplication een aantal duplicaten hebben weggewerkt. Het effect hiervan op de score is echter beperkt, aangezien jullie sinds de eerste upload ook weer nieuw duplicaten geïntroduceerd hebben. Kijk bijvoorbeeld naar Bro.java en Suricata.java, of naar AbstractThreadTool.java en AbstractThreadManager.java.

Bij Unit Size zien we iets soortgelijks: jullie hebben de genoemde voorbeelden wel weggewerkt, maar er zijn tegelijkertijd ook weer nieuwe, lange units geïntroduceerd.

Bij de testcode is er wel een eenduidig beeld: de groei in de productiecode gaat samen met een stijging in de hoeveelheid testcode.

Op basis van deze observaties kunnen we concluderen dat de aanbevelingen van de vorige evaluatie gedeeltelijk zijn meegenomen in het ontwikkeltraject.

The message above is the original one received from Dennis Bijlsma. For convenience the message has been translated to English at the next page.

B.2. English
In the second upload we can see that the code volume has doubled in size, while the score for maintainability stayed about the same. Thus your score is still 4 stars.

On the level of subscores we can see that you cleared a few duplications in the Duplication of Code. The effect of this on your score howeve is limited. Since your first upload you have again introduced more duplication. Look for example at Bro.java and Suricata.java, or at AbstractThreadTool.java and AbstractThreadManager.java.

In Unit Size we can see something similar: You have cleared the before mentioned examples, but at the same time again new, large units have been introduced.

At the test code there is a clear picture though: the increase in the production of code goes hand in hand with the increase of the amount of test code.

Based on these observations we can conclude that the recommendations of the previous evaluation have been partially taken into account in the development process.
The following pages of this appendix will contain some figures that elucidate on the system component architecture of the DokterService. Chapter 6 elaborates textually further on the provided graphics.
Figure C.1: Software Component Architecture on IDS Services
Figure C.3: Software Component Architecture on Web Routing
Figure C.4: Network Analysis Diagram
**BRO AND SURICATA OUTPUT FIELDS**

<table>
<thead>
<tr>
<th>timestamp</th>
<th>rev</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>pcap_cnt</td>
<td>signature</td>
<td>protocol</td>
</tr>
<tr>
<td>event_type</td>
<td>category</td>
<td>http_method</td>
</tr>
<tr>
<td>src_ip</td>
<td>severity</td>
<td>http_refer</td>
</tr>
<tr>
<td>src_port</td>
<td>hostname</td>
<td>type</td>
</tr>
<tr>
<td>dest_ip</td>
<td>url</td>
<td>id</td>
</tr>
<tr>
<td>dest_port</td>
<td>http_user_agent</td>
<td>rrtype</td>
</tr>
<tr>
<td>proto</td>
<td>http_content_type</td>
<td>rrtype</td>
</tr>
<tr>
<td>action</td>
<td>cookie</td>
<td>rrdatal</td>
</tr>
<tr>
<td>gid</td>
<td>length</td>
<td>ttl</td>
</tr>
<tr>
<td>signature_id</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table D.1: Suricata fields
<table>
<thead>
<tr>
<th>Field</th>
<th>Bro Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ts</td>
<td>tx_hosts</td>
<td>section_names</td>
</tr>
<tr>
<td>uid</td>
<td>rx_hosts</td>
<td>version</td>
</tr>
<tr>
<td>id.orig_h</td>
<td>conn_uids</td>
<td>cipher</td>
</tr>
<tr>
<td>id.orig_p</td>
<td>source</td>
<td>curve</td>
</tr>
<tr>
<td>id.resp_h</td>
<td>depth</td>
<td>server_name</td>
</tr>
<tr>
<td>id.resp_p</td>
<td>analyzers</td>
<td>resumed</td>
</tr>
<tr>
<td>proto</td>
<td>mime_type</td>
<td>established</td>
</tr>
<tr>
<td>service</td>
<td>trans_depth</td>
<td>cert_chain_fuids</td>
</tr>
<tr>
<td>duration</td>
<td>method</td>
<td>client_cert_chain_fuids</td>
</tr>
<tr>
<td>orig_bytes</td>
<td>host</td>
<td>subject</td>
</tr>
<tr>
<td>resp_bytes</td>
<td>uri</td>
<td>issuer</td>
</tr>
<tr>
<td>conn_state</td>
<td>user_agent</td>
<td>name</td>
</tr>
<tr>
<td>missed_bytes</td>
<td>request_body_len</td>
<td>notice</td>
</tr>
<tr>
<td>history</td>
<td>response_body_len</td>
<td>peer</td>
</tr>
<tr>
<td>orig_pkts</td>
<td>status_code</td>
<td>certificate.version</td>
</tr>
<tr>
<td>orig_ip_bytes</td>
<td>status_msg</td>
<td>certificate.serial</td>
</tr>
<tr>
<td>resp_pkts</td>
<td>tags</td>
<td>certificate.subject</td>
</tr>
<tr>
<td>resp_ip_bytes</td>
<td>resp_fuids</td>
<td>certificate.issuer</td>
</tr>
<tr>
<td>tunnel_parents</td>
<td>resp_mime_types</td>
<td>certificate.not_valid_before</td>
</tr>
<tr>
<td>mac</td>
<td>node</td>
<td>certificate.not_valid_after</td>
</tr>
<tr>
<td>assigned-ip</td>
<td>filter</td>
<td>certificate.key_alg</td>
</tr>
<tr>
<td>lease_time</td>
<td>init</td>
<td>certificate.sig_alg</td>
</tr>
<tr>
<td>trans_id</td>
<td>success</td>
<td>certificate.key_type</td>
</tr>
<tr>
<td>query</td>
<td>id</td>
<td>certificate.key_length</td>
</tr>
<tr>
<td>qclass</td>
<td>machine</td>
<td>certificate.exponent</td>
</tr>
<tr>
<td>qclass_name</td>
<td>compile_ts</td>
<td>san_dns</td>
</tr>
<tr>
<td>qtype</td>
<td>os</td>
<td>basic_constraints.ca</td>
</tr>
<tr>
<td>qtype_name</td>
<td>subsystem</td>
<td>TC</td>
</tr>
<tr>
<td>rcode</td>
<td>is_exe</td>
<td>referrer</td>
</tr>
<tr>
<td>rcode_name</td>
<td>is_64bit</td>
<td>filename</td>
</tr>
<tr>
<td>AA</td>
<td>uses_aslr</td>
<td>is_orig</td>
</tr>
<tr>
<td>RD</td>
<td>uses_dep</td>
<td>seen_bytes</td>
</tr>
<tr>
<td>RA</td>
<td>uses_code_integrity</td>
<td>total_bytes</td>
</tr>
<tr>
<td>Z</td>
<td>uses_seh</td>
<td>missing_bytes</td>
</tr>
<tr>
<td>answers</td>
<td>has_import_table</td>
<td>timedout</td>
</tr>
<tr>
<td>TTLs</td>
<td>has_export_table</td>
<td>md5</td>
</tr>
<tr>
<td>rejected</td>
<td>has_cert_table</td>
<td>sha1</td>
</tr>
<tr>
<td>fluid</td>
<td>has_debug_data</td>
<td>addl</td>
</tr>
</tbody>
</table>

Table D.2: Bro fields
<table>
<thead>
<tr>
<th>General Document</th>
<th>Bro</th>
<th>Suricata</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timestamp</td>
<td>ts</td>
<td>timestamp</td>
<td>Timestamp associated with</td>
</tr>
<tr>
<td>Source IP</td>
<td>id.orig_h</td>
<td>src_ip</td>
<td>Source IP</td>
</tr>
<tr>
<td>Source Port</td>
<td>id.orig_p</td>
<td>src_port</td>
<td>Source port</td>
</tr>
<tr>
<td>Destination IP</td>
<td>id.resp_h</td>
<td>dest_ip</td>
<td>Destination IP</td>
</tr>
<tr>
<td>Destination Port</td>
<td>id.resp_p</td>
<td>dest_port</td>
<td>Destination port</td>
</tr>
<tr>
<td>Protocol</td>
<td>proto</td>
<td>proto</td>
<td>Protocol</td>
</tr>
<tr>
<td>Event</td>
<td>event_type</td>
<td></td>
<td>Type of event, (DNS, HTTP, fileinfo, alert)</td>
</tr>
<tr>
<td>Identification of application protocol</td>
<td>service</td>
<td></td>
<td>An identification of an application protocol being sent over the connection</td>
</tr>
<tr>
<td>Sent payload bytes</td>
<td>orig_bytes</td>
<td></td>
<td>The number of payload bytes the source sent</td>
</tr>
<tr>
<td>Received payload bytes</td>
<td>resp_bytes</td>
<td></td>
<td>The number of payload bytes the destination sent</td>
</tr>
<tr>
<td>Source Packets</td>
<td>orig_pkts</td>
<td></td>
<td>Number of packets that the source sent</td>
</tr>
<tr>
<td>Source IP Bytes</td>
<td>orig_ip_bytes</td>
<td></td>
<td>Number of IP bytes that the source sent</td>
</tr>
<tr>
<td>Destination Packets</td>
<td>resp_pkts</td>
<td></td>
<td>Number of packets that the destination sent</td>
</tr>
<tr>
<td>Destination IP Bytes</td>
<td>resp_ip_bytes</td>
<td></td>
<td>Number of IP bytes that the destination sent</td>
</tr>
<tr>
<td>Client IP</td>
<td>assigned_ip</td>
<td></td>
<td>Client's actual assigned IP address</td>
</tr>
<tr>
<td>DNS Query</td>
<td>query</td>
<td></td>
<td>The domain name that is the subject of the DNS query</td>
</tr>
<tr>
<td>Query Class</td>
<td>qclass_name</td>
<td></td>
<td>A descriptive name for the class of the query</td>
</tr>
<tr>
<td>Type of query</td>
<td>qtype_name</td>
<td></td>
<td>A descriptive name for the type of the query</td>
</tr>
<tr>
<td>Response code value</td>
<td>rcode_name</td>
<td></td>
<td>A descriptive name for the response code value</td>
</tr>
<tr>
<td>Time To Live</td>
<td>ttl</td>
<td></td>
<td>Time to Live for this record</td>
</tr>
<tr>
<td>Rejected by server</td>
<td>rejected</td>
<td></td>
<td>If the DNS query was rejected by the server</td>
</tr>
<tr>
<td>Resource Record Name</td>
<td>rrname</td>
<td></td>
<td>Resource Record Name (ex domain name)</td>
</tr>
<tr>
<td>Resource Data</td>
<td>rdata</td>
<td></td>
<td>Resource Data (ex IP domain name)</td>
</tr>
<tr>
<td>Filename</td>
<td>filename</td>
<td></td>
<td>A filename for the file if one is available from the source for the file</td>
</tr>
<tr>
<td>Sourced hosts</td>
<td>tx_hosts</td>
<td></td>
<td>If this file was transferred over a network connection this should show the host or hosts that the data sourced from</td>
</tr>
<tr>
<td>Destination hosts</td>
<td>rx_hosts</td>
<td></td>
<td>If this file was transferred over a network connection this should show the host or hosts that the data traveled to</td>
</tr>
<tr>
<td>File source</td>
<td>source</td>
<td></td>
<td>An identification of the source of the file data. E.g. it may be a network protocol over which it was transferred, or a local file path which was read, or some other input source</td>
</tr>
<tr>
<td>HTTP Method</td>
<td>method</td>
<td>http_method</td>
<td>Verb used in the HTTP request (GET, POST, HEAD, etc.)</td>
</tr>
<tr>
<td>HTTP User Agent</td>
<td>user_agent</td>
<td>http_user_agent</td>
<td>Value of the User-Agent header from the client</td>
</tr>
<tr>
<td>HTTP Referer</td>
<td>referrer</td>
<td>http_refer</td>
<td>Value of the referer header</td>
</tr>
<tr>
<td>HTTP Protocol</td>
<td>protocol</td>
<td>protocol</td>
<td>protocol used by the HTTP</td>
</tr>
<tr>
<td>Hostname</td>
<td>host</td>
<td>hostname</td>
<td>Value of the HOST header</td>
</tr>
<tr>
<td>URL</td>
<td>uri</td>
<td>url</td>
<td>URL at hostname accessed/URI used in the request</td>
</tr>
</tbody>
</table>

Table D.3: Fields shown in the settings-section of the General Report.
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request body length</td>
<td><code>req_body_len</code> Actual uncompressed content size of the data transferred from the client</td>
</tr>
<tr>
<td>Response body length</td>
<td><code>rest_body_len</code> Actual uncompressed content size of the data transferred from the server</td>
</tr>
<tr>
<td>Alert Signature</td>
<td><code>signature</code> Signature string associated with the alert</td>
</tr>
<tr>
<td>Alert Category</td>
<td><code>category</code> Category of the alert</td>
</tr>
<tr>
<td>Alert Severity</td>
<td><code>severity</code> The danger of the alert</td>
</tr>
<tr>
<td>Operating System</td>
<td><code>os</code> Operating system</td>
</tr>
<tr>
<td>Is .exe</td>
<td><code>is_exe</code> If the file is executable</td>
</tr>
<tr>
<td>Server name</td>
<td><code>server_name</code> Value of the Server Name Indicator SSL/TLS extension. It indicates the server name that the client was requesting</td>
</tr>
<tr>
<td>The weird name</td>
<td><code>name</code> The name of the weird that occurred</td>
</tr>
<tr>
<td>Additional weird information</td>
<td><code>addl</code> Additional information accompanying the weird if any</td>
</tr>
<tr>
<td>Origin country</td>
<td><code>orig_cc</code> The origin country of the packet</td>
</tr>
<tr>
<td>Destination country</td>
<td><code>resp_cc</code> The destination country of the packet</td>
</tr>
<tr>
<td>Alert information</td>
<td><code>event_type = alert</code> Alert information</td>
</tr>
<tr>
<td>DNS information</td>
<td><code>event_type = dns</code> DNS information</td>
</tr>
<tr>
<td>TLS information</td>
<td><code>event_type = tls</code> TLS information</td>
</tr>
<tr>
<td>HTTP information</td>
<td><code>event_type = http</code> HTTP information</td>
</tr>
<tr>
<td>File information</td>
<td><code>event_type = fileinfo</code> File information</td>
</tr>
</tbody>
</table>

Table D.4: Fields used for the Malicious Report.
"The Info Sheet is contained on the next page."
GENERAL DESCRIPTION OF PROJECT DOKTERSERVICE

Title of the project: Dokterservice: A Security Information and Event Management System
Name of the client organization: KPMG
Date of the final presentation: June 26, 2015

DESCRIPTION
KPMG is a multinational firm, which is focused on three pillars: tax advisory, accountancy and advisory. The advisory section, in particular the IT Advisory, required a system that is able to analyze, process and visualize captured network traffic. This network traffic is gathered from the own company or their clients. The goal of this project is to design and develop a "Security Information and Event Management System (SEAMS)" that manages Network Intrusion Detection services and provides a report specifying the condition of the analyzed network. In order to manage the different Intrusion Detection Systems (IDS) their results need to be merged. Every IDS produces a different result based on their focus. To gain the most valuable information of the network traffic, one has to merge all IDS's results. The major challenge in developing such a SEAM System is that the system should be able to process an enormous amount of data. This data is roughly between 2-10TB. The stakeholder required the system to be built in a modular way. This leads to both maintainability and adaptability of the system since the customer accounts for the extension of the product. During the research the development team acquired a large amount of knowledge about network traffic. In addition to this knowledge, they gained understanding and expertise in the used IDSs. By gathering this knowledge, the product got molded into a useful, extendable and efficient Dokterservice for the client. The process, for the development of the Dokterservice, was set up by means of the Scrum methodology. A change of requirements is a pristine reason to apply the scrum methodology, as development tasks are flexibly generated. Hence during the development process, any (un)expected changes could easily be adapted to the product and into Scrum.

The dokterservice that is created is able to read, process, merge and visualize the data gathered from the network traffic. Continuously throughout the development, the product got tested on captured malicious network traffic, which varied tremendously in size. Furthermore, the product got tested by means of unit tests and continuous integration. This ensured the team that the developed components work as they should. In addition to the aforementioned ways of testing, we also had continuous interaction with our stakeholders and organized demo's for possible future end-users. The future outlook of the Dokterservice is very prosperous. KPMG plans on applying the Dokterservice for future network analysis. We also have provided some future work recommendations which are included in our final report.

TEAM MEMBERS, CLIENT, COACH AND COMMITTEE MEMBERS

Name: Mourad el Maouchi
Interests: Cyber Security
Role & contribution: Product owner & Back- & front-end developer
contact: mouradelmaouchi.com

Name: Ricky Sewsingh
Interests: Software Development
Role & contribution: Back-end developer
contact: r.sewsingh218gmail.com

Name: Wesley van der Lee
Interests: Cyber Security
Role & contribution: Scrum Master & Back- & front-end developer
contact: wesleyvdlee@gmail.com

Name: Aidan Mauricio
Interests: Software development & Security
Role & contribution: Back-end developer
contact: a.c.a.mauricio47@gmail.com

Clients: M. Hovens & D. Wagenaar (KPMG IT Advisory)
Coach: Dr. C. Doerr (Department of Intelligent Systems, TU Delft)
Contact regarding project: Wesley van der Lee (wesleyvdlee@gmail.com) & Mourad el Maouchi (mourad@elmaouchi.com)

The final report for this project can be found at: http://repository.tudelft.nl
Final Requirements

1. Data input consists of multiple .pcap-files that contains captured network traffic.
2. .pcap-files are parsed with the original timestamps.
3. The micro instance is required to be mounted to a specific static directory. This is user instantiated by terminal commands.
4. The information stored in a database should be in application independent format.
5. The database should scale well for large amounts of data (> 10 GB).
6. The application should be build in a modular way, allowing addition and removal of modules. (e.g. different modules for the different functionality, extra NIDSs).
7. Computation time should not exceed 2 days (48 hours).
8. The application is web-based. No command line interaction should be necessary to upload a .pcap-file, generate a report or view any retrieved data.
9. The access to the web-based application is properly authenticated.
10. Malware analysis is done with at least:
    (a) Bro
    (b) Suricata
11. The rule list is updated prior to the start of an analysis.
12. Automatic file carving (i.e potential malicious files are carved from the traffic and stored on disc) (Optional)
14. The analysis report should contain:
    • Top X lists, where X can be set via the user interface by the user.
    • Top X lists, where a user can determine via the user interface if the list should be ordered ascending or descending.
    • Top X lists of at least:
      – Most visited countries
      – Most visited IPs
      – Most used protocols
      – Most used ports
      – Most origination IPs
F. Final Requirements

- Most visited URLs
- Largest files downloaded
- Most used User Agents (HTTP(S) traffic)

- Via the user interface, a user should be able include or exclude other fields.
- List of malware that was found (based on Suricata output; e.g. Critical / High alerts), which should be distinguish based on Suricata-output-type. The user must be able to filter these results.
- Visualize actual packet data with a certain time margin from the .pcap-file with CapMe. (Optional)

15. Output formats of the report should contain:

- Table contents in the reports should be downloadable in a CSV-format.
- Generated charts should be downloadable in a PNG format.
- A world map colored according to the amount of source and destination IP’s originating from the corresponding country.
The following table describes our final timeline as executed.

<table>
<thead>
<tr>
<th>Week</th>
<th>Iteration</th>
<th>Milestone/Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>N.A.</td>
<td>· Project Demarcation</td>
</tr>
<tr>
<td>2</td>
<td>N.A.</td>
<td>· Research</td>
</tr>
</tbody>
</table>
| 3    | #1        | · Research       
|      |           | · Program Outline |
|      |           | · Basic Skeleton of Interacting Components |
| 4    | #2        | · Research 2     |
| 5    | #3        | · New Basic Skeleton of Interacting Components |
|      |           | · Managers [NetworkToolManager, Manager] |
|      |           | · MongoDB        |
| 6    | #4        | · Bro            
|      |           | · Suricata       |
|      |           | · NetworkTool    |
|      |           | · Web Interface 1 |
| 7    | #5        | · General Document |
|      |           | · Report         |
| 8    | #6        | · Authentication 
|      |           | · Web Interface 2 |
|      |           | · Report         |
|      |           | · Network Analysis |
| 9    | #7        | · Formatted output |
|      |           | · Server Testing |
| 10   | #8        | · Finishing up the product |
|      |           | · Preparing for the final presentation |

Table G.1: Final Project Timeline.


