Real Time Threat Detection Through Network Analysis

Bachelor Thesis Report

by

Kabilan Gnanavarothayan
Pravesh Moelchand
Just van Stam
Jim Verheijde

to fulfil the requirements for obtaining the degree of

Bachelor of Science
in Computer Science and Engineering

at the Delft University of Technology,
to be defended publicly on Tuesday July 2, 2019 at 15:00.

Project duration:	April 23, 2019 – June 28, 2019	
Thesis committee:	Otto Visser	TU Delft, Bachelor Project Coordinator
	Huijuan Wang	TU Delft, Bachelor Project Coordinator
	Harm Griffioen	TU Delft, Bachelor Project Coach
	René Kalff	Intermax Cloudsourcing B.V.

TU Delft Delft University of Technology

Summary

Intermax Cloudsourcing B.V. designs, implements and manages critical IT-infrastructures for Dutch clients from the medical, public and financial sectors. The information that passes over these IT-infrastructures is highly confidential and privacy-sensitive, therefore it is essential that these infrastructures are secure. To improve the security of their infrastructure, Intermax is developing a Security Operations Center (SOC) which is fed by information from an optical tap which is placed on one of Intermax's core routers.

The goal of this project was to extend the SOC with a REST API that analyses and classifies SSL certificates to filter malicious network data. This REST API makes use of models, so in addition to the REST API itself, a Neural Network Framework has been built to create these models. The framework can be used for different sorts of network data, but for this project, a proof of concept using SSL certificates was worked out to provide Intermax with a working product.

The SOC is being built upon the security analytics framework Apache Metron and the REST API will be incorporated using Metron's Model as a Service functionality. Apache Metron sends individual data packets to the REST API, which analyses the data packet and returns whether the packet is malicious or not with a certain accuracy. This analysis takes roughly 1 millisecond and is independent from other data packets. This allows Apache Metron to spawn multiple instances of the REST API, making the solution fast and scalable.

The Neural Network Framework runs completely separate from Apache Metron and the REST API. A user can configure, train and test a neural network using the framework and their own dataset. The neural network can be stored on a storage medium. Consequently, the REST API can import and apply the neural network on incoming data.

The process of creating the product was different than expected. Throughout the Bachelor, Scrum was used for project development, but during the development phase, it became clear that Kanban was a better fit for this project. Next to that, the research phase took almost twice as long as expected. This was because the initial solution could be implemented easily using Apache Metron's functionality, leaving too little space for decent software engineering. The project was adjusted and this cost the team extra research time. Nevertheless, the team and Intermax are content with the system that has been delivered in the short amount of time.

Preface

This project had many obstacles throughout its development, which we would not have overcome without the help of certain individuals. Firstly, we would like to thank Daniel Pot, who was our first contact from Intermax and helped us establish the initial project. Secondly, we would like to thank our supervisor René Kalff for helping us throughout the challenges we faced when we had to develop and deploy our framework. Next, we would like to thank our coach Harm Griffioen and Otto Visser for making sure that our project lived up to the TU Delft standards and gave us more insight to our problem. Finally, we would like to thank all the people at Intermax who created a great atmosphere to work in and of course the ladies of Intermax hospitality for the delicious food.

Kabilan Gnanavarothayan
Pravesh Moelchand
Just van Stam
Jim Verheijde
Rotterdam, June 2019

Contents

Summary i

Preface ii

List of Figures vi

List of Tables vii

1 Introduction 1
1.1 Context . . . 1
1.2 Structure . . . 1

2 Problem definition 2

3 Problem analysis 3
3.1 Changing project descriptions . . . 3
3.2 Intermax's requirements . . . 3
3.3 TU Delft's requirements . . . 3
3.4 Final requirements . . . 4
3.4.1 Functional requirements . . . 4
3.4.2 Non-Functional requirements . . . 5

4 Problem Solution 6
4.1 SSL certificate classification . . . 6
4.2 Neural Network introduction . . . 6
4.3 Framework . . . 7
4.4 Existing solutions . . . 7
4.5 Conclusion . . . 8

5 Development process 9
5.1 Planning . . . 9
5.2 Development methodology . . . 9
5.3 Internal communication and agreements . . . 10
5.4 External communication . . . 10
5.5 Version control . . . 10
5.6 Other tools . . . 10

6 Architecture design 11
6.1 Apache Metron . . . 11
6.1.1 Sensors . . . 12
6.1.2 Kafka . . . 12
6.1.3 Storm . . . 12
6.1.4 Hadoop . . . 12
6.1.5 Pipeline . . . 12
6.1.6 MaaS . . . 13
6.2 Neural Network Framework . . . 14
6.2.1 Command line interface . . . 14
6.2.2 Configuration . . . 14
6.2.3 Neural network . . . 15
6.3 Workflow and data flow . . . 16
6.3.1 User perspective . . . 16
6.3.2 Data perspective . . . 17

7 Programming languages and Frameworks **18**
7.1 Programming languages . . . 18
7.1.1 Pros and Cons . . . 18
7.2 Frameworks . . . 18
7.2.1 REST API . . . 19
7.2.2 Machine learning framework . . . 19

8 System usage **21**
8.1 Neural Network Framework . . . 21
8.1.1 Neural network . . . 21
8.1.2 Command Line Interface . . . 21
8.1.3 Data import . . . 22
8.1.4 Configuration . . . 22
8.1.5 Export . . . 22
8.2 REST API . . . 23
8.2.1 Model . . . 23
8.2.2 Configuration . . . 23
8.2.3 Deployment . . . 25

9 Proof of concept **26**
9.1 Data selection . . . 26
9.1.1 Uniqueness . . . 27
9.1.2 Data set creation . . . 27
9.2 Feature Selection . . . 27
9.2.1 Normalisation . . . 27
9.2.2 43 features . . . 28
9.2.3 7 and 9 features . . . 28
9.3 Neural Network structure . . . 28
9.4 Model Selection . . . 28
9.4.1 Metrics . . . 29
9.4.2 Results . . . 29
9.5 Conclusion . . . 31
9.6 Discussion . . . 31

10 Validation and Verification **33**
10.1 Validation . . . 33
10.1.1 Functional Requirements satisfaction . . . 33
10.1.2 Use case: C2 communication SSL certificate classification . . . 34
10.1.3 Neural Network Framework usability . . . 34
10.2 Verification . . . 35
10.2.1 Neural Network Framework Verification . . . 35
10.2.2 REST API verification . . . 35
10.2.3 System verification . . . 35
10.2.4 Scalability and Speed . . . 35
10.2.5 Non-functional requirements satisfaction . . . 36
10.2.6 Testing . . . 36
10.2.7 Software Improvement Group Code Quality Assessment . . . 37

11 Ethical considerations **38**

12 Discussion **39**
12.1 Tensorflow . . . 39
12.2 Metron . . . 39
12.3 Development process . . . 40
12.4 Improvements . . . 40

13 Conclusion **41**

14 Future Work **42**

A Research report **43**
A.1 Introduction . . . 43
A.2 Research phase 1 . . . 43
A.2.1 Client requirements . . . 43
A.2.2 Threat detection methods . . . 43
A.2.3 Apache Metron and reconsideration . . . 44
A.3 Research phase 2 . . . 45
A.3.1 TU Delft software requirements . . . 45
A.3.2 Adapted client requirements . . . 45
A.3.3 Possible solutions . . . 45

B Overview of possible solutions **47**
B.1 Anomaly detection . . . 47
B.2 Misuse detection . . . 48

C Project Planning **49**

D Manuals **51**

E Configuration files **55**
E.1 SSL certificate model . . . 55
E.1.1 Neural Network Framework . . . 55
E.1.2 REST API . . . 56

F Original project description **59**

G List of SSL features **61**

H Weekly logs **63**
H.1 Process overview . . . 63
H.1.1 Week 1 (22-04-2019) . . . 63
H.1.2 Week 2 (29-04-2019) . . . 63
H.1.3 Week 3 (06-05-2019) . . . 63
H.1.4 Week 4 (13-05-2019) . . . 63
H.1.5 Week 5 (20-05-2019) . . . 64
H.1.6 Week 6 (27-05-2019) . . . 64
H.1.7 Week 7 (03-06-2019) . . . 64
H.1.8 Week 8 (10-06-2019) . . . 64
H.1.9 Week 9 (17-06-2019) . . . 64

I Software Improvement Group Code Quality Evaluation **65**
I.1 Feedback REST API . . . 65
I.2 Feedback Neural Network Framework . . . 66

J Measurements **67**

Bibliography **68**

List of Figures

4.1 A simple neural network with 3 neurons in the input layer, 2 hidden layers with both 4 neurons, and 1 neuron in the output layer 7

6.1 Global architecture diagram 11
6.2 Metron architecture 13
6.3 Neural Network Framework diagram 16

8.1 Example of export directory structure 23

9.1 Precision recall curve for the [25, 20, 15, 10] model with 9 normalised features . . 31

List of Tables

7.1 Pros and cons of different programming languages that could be used 18

9.1 Number of certificates collected 26
9.2 Neural Network performance comparison for the phishing and command and control certificates on 43 features 29
9.3 Neural Network performance comparison for the phishing certificates on 7 normalized, 9 and 9 normalized features 30
9.4 Neural Network performance comparison, note that the model highlighted in green is the chosen model for this proof of concept 30

10.1 Must have requirements satisfaction 33
10.2 Should have requirements satisfaction 33
10.3 Could have requirements satisfaction 34
10.4 Table showing the average, median and standard deviation of the request time to the REST API server in milliseconds, see Table J.1 for original data 36
10.5 Non-functional requirements satisfaction 36

G.1 Features from the Torroledo et al. study[25] 61
G.2 Table of 43 feature used in the first models 62
G.3 Table of 7 and 9 features 62

J.1 The measurements of the REST API server request delay in milliseconds 67

1 Introduction

1.1. Context

In the current day and age, internet communication is essential to businesses and individuals. Intermax Cloudsourcing B.V. designs, implements and manages critical IT-infrastructures for Dutch clients from the medical, public and financial sectors. The information that passes over these IT-infrastructures is highly confidential, therefore it is essential that these infrastructures are secure. In addition to providing safe and reliable infrastructures, Intermax is developing a Security Operations Center (SOC). The goal of this SOC is to analyse the data from the IT-infrastructures to detect security incidents and to enable data-driven decision making.

This project will extend the SOC with a REST API which will analyse and classify network data, in particular SSL certificates, coming from an optical tap placed by Intermax. In addition to the REST API, a Neural Network Framework has been created to train models which can in turn be used by the REST API.

1.2. Structure

The report is structured as follows: in chapter 2, the problem is further explained and a problem definition is given. Successively, chapter 3 goes deeper into the problem and derives the requirements for the project. From these, the solution to the problem is proposed in chapter 4. The process of developing this solution is laid out in chapter 5. In chapter 6, the architecture design of the proposed system is presented. The programming languages and frameworks that have been used are elaborated in chapter 7. Subsequently, chapter 8 provides an explanation on how to use the system. chapter 9 focuses on the specific use case that was worked out for Intermax using the REST API and Neural Network Framework. In chapter 10 the system is validated against the requirements and its functionality is verified. chapter 11 reviews the ethical considerations of the system. After that, chapter 12 discusses the challenges that the team encountered and possible improvements. Finally, chapter 13 and chapter 14 conclude the report with a conclusion and recommendations for future work.

2

Problem definition

Intermax is developing a SOC to improve the safety and reliability of their infrastructures. To this end, Intermax placed an optical tap on one of their core routers. From this tap, all the connections that are made by Intermax and its clients with addresses outside the network are sent to the SOC. This data stream can reach speeds up to 500 Mbit/s. The SOC is based on Apache Metron[1], which is a security analytics framework, and will incorporate a REST API using its Model as a Service (MaaS)[2] infrastructure.

The goal of this project is to create the REST API, which will analyse the data and classify it. The REST API will do this using models that will be created using a framework that will also be created within the scope of this project. Intermax wants the team to deliver a proof of concept of the REST API that can classify malicious and benign SSL certificates. The project is successful if a model can be created with the framework and this model is applied successfully by the REST API on the SSL certificate proof of concept. This system should also be scalable to be able to handle the possible 500 Mbit/s data stream.

[1] http://metron.apache.org/
[2] https://metron.apache.org/current-book/metron-analytics/metron-maas-service/index.html

3

Problem analysis

3.1. Changing project descriptions

The original project description that was posted on BEPSys (see Appendix F) differed from the final problem definition in chapter 2. Intermax originally wanted the team to only create the REST API and corresponding models to classify network data. Upon this, research was done on different network threat detection methods. After two weeks, it became clear that this would not satisfy the requirements of the TU Delft. Thereupon, the project description was reformulated leading to the final problem definition as presented in chapter 2.

It would be a waste to completely disregard the research that was done in the first two weeks, therefore, this research and its results are presented in phase 1 of the research report in section A.2. This research report also contains the research that was done for the final problem definition, this can be found in section A.3.

In the remainder of this chapter, the requirements of Intermax and the TU Delft are described. From these requirements, the final requirements for the software product are derived.

3.2. Intermax's requirements

As the client already has a lot of the infrastructure in place, this imposes several requirements for the software product that is going to be designed. As mentioned before, the SOC that is being developed by Intermax is being built upon Apache Metron. Intermax therefore requires the software product to be a REST API that will be incorporated into Apache Metron through its MaaS functionality. More about this in chapter 6.

Furthermore, the data that is being provided through the optical tap is not allowed to leave Intermax's network due to confidentiality constraints. In addition, this data will only consist of traffic that is entering or leaving Intermax's network, so no internal network traffic is available.

Intermax stresses that they want to see a working prototype at the end of the project. Therefore, they would like to have a proof of concept of the REST API which uses a model that is focused on classifying SSL certificates to detect command and control communication. Which is communication that "is used to instruct compromised machines to perform malicious activity". [8] This prototype does, however, not yet have to work with the 500 Mbit/s data stream, but it should be scalable.

3.3. TU Delft's requirements

The TU Delft requires that a certain level of software engineering is applied during the Bachelor End Project. During the research phase, it became clear that the original functionality that Intermax desired, could easily be implemented by using existing functionality within Apache Metron. This would mean that the required level of software engineering would not be satisfied. This problem was discussed with the Bachelor End Project coordinator, Otto

Visser, and it was indicated that creating a framework around the creation of models would suffice the requirements of the TU Delft. To this end, the software product will consist of a framework to train models and a REST API to apply these models on the incoming network data.

3.4. Final requirements

The final requirements have been composed from the requirements of the client and the requirements of the TU Delft. They will be presented using the MoSCoW method [1] in this section, divided into functional and non-functional requirements.

3.4.1. Functional requirements

Must have

- The framework must be able to generate a model from a given data set
- The framework must be able to give accurate predictions based on the input data
- The framework must be able to provide a classification model
- The framework must be able to export the generated model
- The user must be able to specify the input data and neural network configuration through a config file
- The user must be able to start the framework through a command line interface (CLI)
- The user must be able to specify the config file location for the framework
- The API must be able to import the generated model
- The API must be deployable by Metron
- The API must be able to return an error message if incorrect input data is passed through
- The API must return a classification of the input data
- The API must run consistently with Metron without crashing
- An API instance must be independent of other API instances
- The user must be able to specify the model location through a config file
- The user must be able to specify the config file location for the REST API

Should have

- The classification should include an accuracy score
- The API should not delay the data stream significantly
- The API should be able to transform input data to neural network features using custom code
- The user should be able to verify the model using specified test data
- The user should be able to see when the training of the model is done
- The user should be able to decide whether the model should be exported or not
- The user should be able to specify the input transform function for the API through a config file

Could have

- The framework could output benchmarks
- The user could be able to specify whether error message are returned
- The user could be able to see a progress bar of training the model
- The user could be able to specify a port number in the config file

Won't have

- The system will not detect DDOS, probing and port scanning attacks.
- The system will not decrypt actual data
- The API will not send data to external sources
- The framework and API will not directly communicate with eachother

3.4.2. Non-Functional requirements

- The system must be incorporated into Apache Metron as a REST API
- The API should be able to run on Linux
- There should be a user manual for the framework
- There should be a user manual for the API
- The testable code should have a branch coverage of >75% (scripts/scrapers are not part of the main software engineering project so those are marked as non testable)

4 Problem Solution

In this chapter, descriptions will follow on how the problem, based on the requirements set by both Intermax and TU Delft, is going to be tackled. Afterwards, an overview of the existing solutions will be given. Followed by the conclusion, which will give an overview of our solution and how it solves the problem as defined in chapter 2.

4.1. SSL certificate classification

Based on the research that was done in the research report (Appendix A), a neural network is going to be used to detect SSL certificates used in command and control communication. A neural network approach will be used, as it can also classify previously unseen certificates. Other solutions like blacklists were not appropriate because it is already implementable within Apache Metron. This neural network would then have to be deployed as a Model as a Service (MaaS) within Metron. This solves the problem as pointed out by Intermax where they want to detect SSL certificates used for command and control traffic.

4.2. Neural Network introduction

In case the reader has no prior knowledge to neural networks this section serves as a very basic introduction to the concept.

A neural network is on the highest level a collection of neurons, which are connected by edges. These directional edges can transmit a signal from one neuron to another, similar to the neurons in a human brain. A neuron that receives a signal processes it and then signals additional neurons connected to it via its edges.

In neural networks, the signal at an edge between neurons is a real number. The output of each neuron is computed by some non-linear function of the sum of the values on the input edges. The way neural networks can learn is by adjusting the weights of the neurons and edges. The weight at an edge determines the strength of the signal that is sent over that edge.

Typically, neurons are aggregated into layers. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing multiple hidden layers. A visualisation of a simple neural network is given in Figure 4.1. In this network the neural network has 3 input neurons, which are real numbers. After the input layer are the two hidden layers consisting of 4 neurons each. Finally the last hidden layer is connected to the output layer consisting of just one neuron. Note that it usually the case that one layer is only connected to the one behind it and in front of it, not two layers ahead or behind. The network is structured this way to make the calculation for the training of the network, namely backpropagation, easier. [9]

As mentioned just before, the network trains using an algorithm called backpropagation. Roughly speaking, this algorithm approximates the optimal weights for the neural network by feeding it the training data and changing the weights based on the error the network

produced on each training sample. The larger the error becomes, the more the weight is punished by lowering it and vice versa. [9]

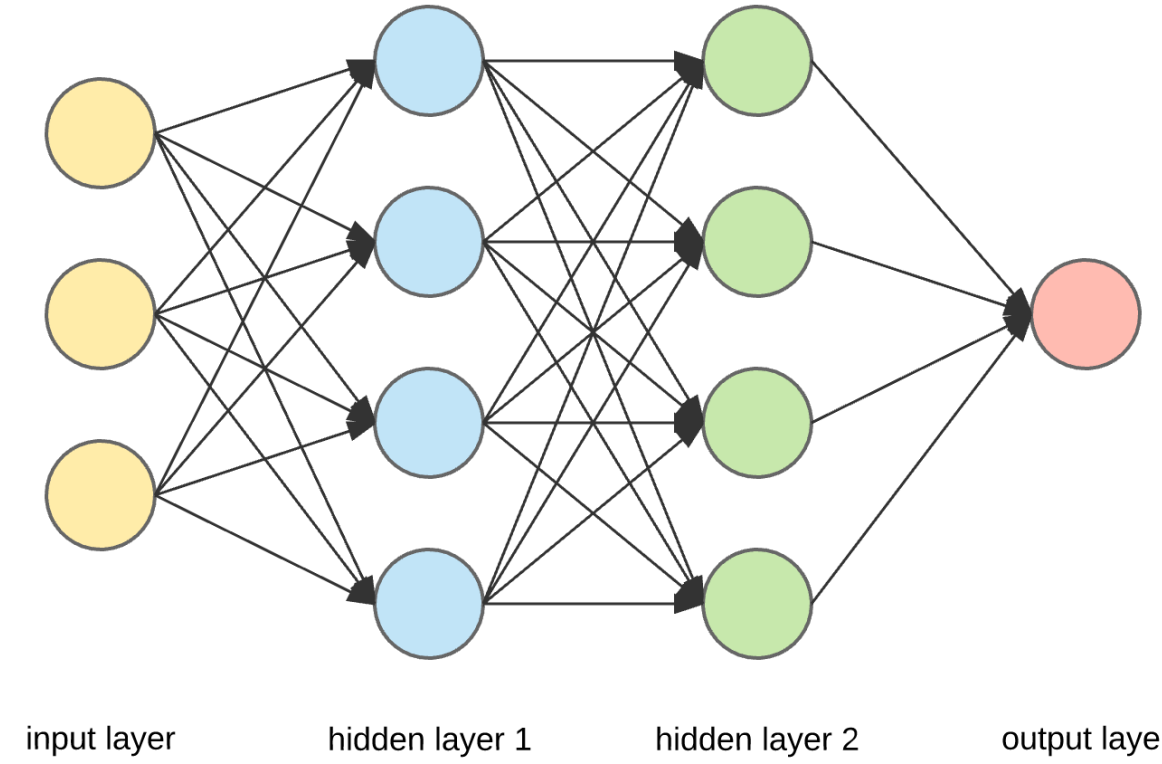

Figure 4.1: A simple neural network with 3 neurons in the input layer, 2 hidden layers with both 4 neurons, and 1 neuron in the output layer

4.3. Framework

The requirements from the TU Delft stated that the team has to make some kind of framework for the SSL classification neural network algorithm in order for the software engineering component to remain significant. As the team was already going to use neural networks for the SSL certificates, it was decided that a more general framework would be made for constructing neural networks. This way a user of this framework can build any neural network they want and immediately use it with Apache Metron. This solves the problem laid out by TU Delft where the team has to maintain a substantial amount of software engineering in the project.

4.4. Existing solutions

There are no existing solutions that solve this specific problem. There exist no tools that integrate Apache Metron with a machine learning framework. The only solutions that come somewhat close to the solution presented here, are tools that put a front end around machine learning frameworks. Examples of this are Splunk[1] or Deep Learning Studio[2]. However these tools are not meant to be very customizable or exportable in any way. They only work in their own environments. Therefore the solution proposed by the team is a unique product that hasn't been developed before.

[1] https://www.splunk.com/en_us/software.html
[2] https://deepcognition.ai/features/deep-learning-studio/

4.5. Conclusion

The final system which will solve the problem will consist of three main components: Apache Metron, the REST API as a MaaS and the Neural Network Framework. The Neural Network Framework will solve TU Delft's requirement of having sufficient software engineering in the end product, because writing a framework involves a lot more software engineering due to the generalisation that has to be implemented in the framework. The system as a whole will solve Intermax's problem of detecting SSL certificates used for command and control communication, because the framework will be able to train a model which can classify SSL certificates as command and control certificate or benign. This model can then be deployed as a MaaS enabling this classification to be done in real time on live network data. Therefore together the three main components together will both solve Intermax's problem of detecting SSL certificates used for command and control communication, and the requirements of the TU Delft regarding the sufficient amount of software engineering.

5 Development process

This chapter will contain a description of the development process of the project. The approach for working as a team and the tools that are used to aid the development of the product will be described.

5.1. Planning

In the beginning of the project a general project planning was made, to make sure that enough progress was made during the project. As it was not yet clear what for different components had to be developed, it was hard to make a detailed planning. The first 2 weeks were dedicated to understand the problem, doing research and coming up with a solution. The next two weeks were scheduled to create an architecture design and setting up the software environments and frameworks. Week 5 to 8 the software product had to be made and tested and the last two weeks were dedicated for the final report and presentation. The complete planning can be found in Appendix C.

5.2. Development methodology

The decision was first made to use Scrum[1] for agile development during the implementation phase of the project. Scrum works with development cycles called sprints. There is a product backlog which contains the tasks that need to be finished for the project. For every sprint, tasks are picked that will be done during the sprint. At the end of each sprint, the process of that week is reviewed in the sprint retrospective. Each sprint will have a duration of one week.

Scrum has been taught throughout the Bachelor as *the way to go* when doing a software project, so every team member has experience with it. All team members are therefore also used to the Scrum workflow, that should allowing for quick development.

In the first week of implementation, the team started with a planning and selected several tasks to finish in the first sprint. These tasks were relatively general as the team did not know yet what specific things had to be done when working with the various frameworks. The team noticed however that new problems were encountered on a daily basis. These problems were, in most cases, solved within a sprint, meaning they were not taken into account in the planning. There were also a lot of problems with the Apache Metron framework, which caused one team member to work on the same task for over 2 weeks in total.

Upon these problems, the team re-evaluated and decided to change the development style and went for a Kanban[2] approach. Kanban allows for a more dynamic workflow as the team focuses on the next problem they encounter. This allowed the team to work on the problems that were encountered daily and did not require them to have a planning at the start of the

[1] https://www.scrum.org/resources/what-is-scrum
[2] https://www.atlassian.com/agile/kanban

week. Making a planning at the start of the week was quite hard because of the many unexpected problems.

For keeping track of the tasks that needed to be done, the GitLab issueboard [3] was going to be used. This was however causing too much overhead, as the new problems encountered could often be solved rather quickly. In addition, the team members were working side by side every day in the Intermax office, so any new issues were discussed face to face, leaving out the need for discussing code online in the GitLab project. However the team is of the opinion that if the size of the team was larger this wouldn't have worked because while one can memorize what the 3 other team members are doing this is not possible for much larger teams.

The sprint retrospectives were replaced with weekly logs of the things that were done that week. The logs were made as a general overview of the development process, and some of the problems that were encountered during the project were described. These logs can be found in Appendix H

5.3. Internal communication and agreements

The team worked daily from the Intermax office in Rotterdam. This meant that most of the communication within the team was done face to face in real life. When not in the office, communication was done through a whatsapp group chat.

The team agreed to work daily at the office. In case of absence, the group member in question would notify the team through the group whatsapp. Everyone in the team agreed to put an equal amount of effort in the project and compensate for missed hours where necessary.

5.4. External communication

Before the team had arranged the project at Intermax, all communication with external parties went through mail. Once the project at Intermax was arranged, the team worked in the Intermax offices and had direct contact with René Kalff, who was the coach from Intermax. René was given regular updates in an informal way when he was present in the office. There has been an additional midterm meetings with Intermax's head of HR and CTO to give them an update on the progress as well.

5.5. Version control

A GitLab instance on the servers of Intermax was used as the version control platform. In this way, the code would remain within the Intermax network. The GitLab CI was used for continuous integration to check whether the code was working well. This CI would also check the code style and the pipeline would fail in the case of style errors. As said, GitLab issues were used in the beginning of the implementation phase, but deemed redundant after some time.

5.6. Other tools

All of the team members used the PyCharm IDE to develop their code. Google Drive was used to store and share the meeting documents, presentations and any other documentation that needed to be shared. Overleaf was used to write this final report together in LaTeX.

[3] https://about.gitlab.com/product/issueboard/

6

Architecture design

From the problem analysis in chapter 3 it became clear that three main components had to be combined: Metron, MaaS and the Neural Network Framework. These systems form the total architecture of our project. Of these three systems, two are developed by the team, namely MaaS and the Neural Network Framework. A diagram of the system is depicted in Figure 6.1. In the sections below these three subsystems will be described individually. Finally the workflow of the combined system is described. Note that choices regarding programming languages and frameworks have not yet been made at this stage. These are made after the architecture design, as one cannot decide what programming language to choose if the system that is to be implemented has not been designed yet.

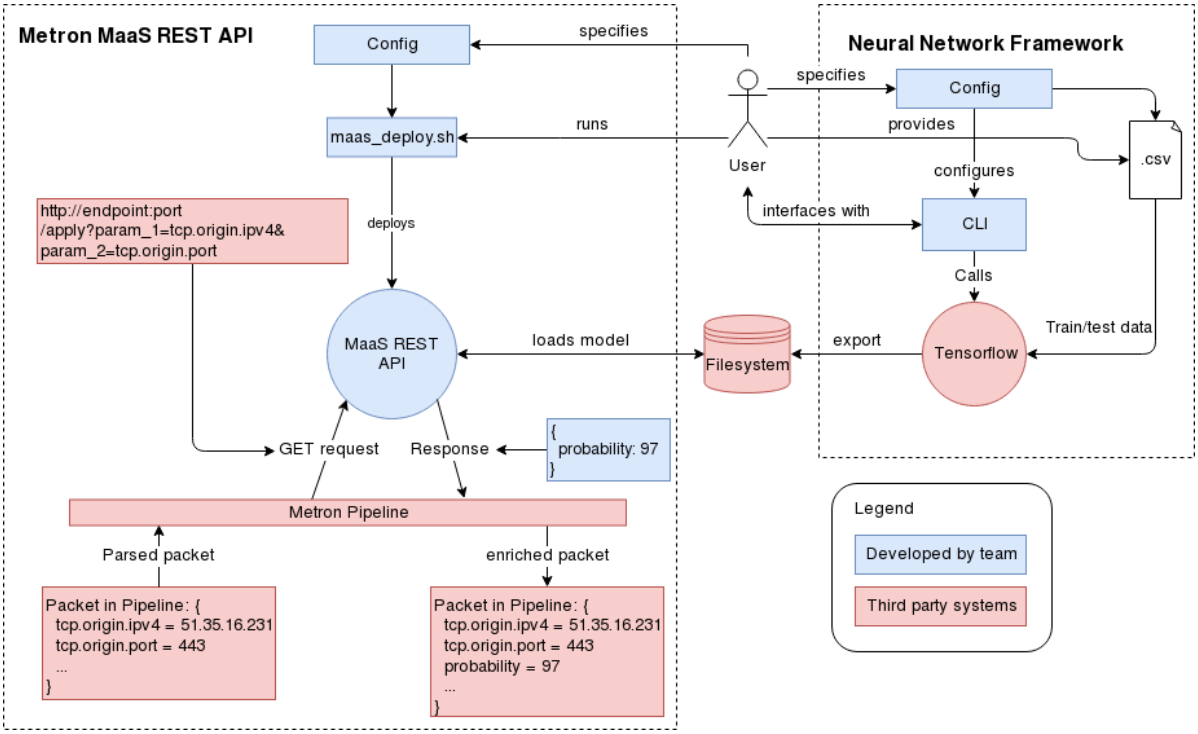

Figure 6.1: Global architecture diagram

6.1. Apache Metron

In order to understand this project it is also important to understand the basics and purpose of Apache Metron. Apache Metron is an open source tool which integrates a variety of other open source big data technologies for security monitoring and analysis. Because of this

variety, Metron's capabilities can be divided in 4 areas, namely as a tool to capture, store and normalize any type of data at extremely high rates, for real time processing and application of enrichments, efficient information storage and an interface for the user to view data and alerts that have passed through the system. The main open source tools that are at the core of Metron are Apache Kafka, Apache Storm and Apache Hadoop. A high level overview diagram of Metron can be seen in Figure 6.2. The different parts of Metron are described below.

6.1.1. Sensors

Sensors in Metron are processors for telemetry sources, they regulate the data stream between telemetry sources and Apache Metron. These telemetry sources can be Network Intrusion Detection Systems (NIDS) or other systems capable of delivering network telemetry. The capabilities differ per telemetry source, for example Zeek is capable of parsing SSL certificates from a raw network stream, whereas other sensors like Squid are capable of parsing other components of raw network streams as well. The sensors themselves are capable of collecting the data from each of these telemetry sources and pushing it into Kafka streams which go into the Metron system for further processing.

These telemetry sources have to be installed and configured first. Then they can be added to Metron by connecting them to the designated sensors. These sensors are regulated in Metron by Storm.

6.1.2. Kafka

The data stream that is collected by the sensors will be sent to Kafka, a stream processing engine. This component of Metron is responsible for the data streams within Metron. Besides the data ingress it is also responsible for data egress in the indexing part of Metron described in subsection 6.1.5

6.1.3. Storm

The application that is responsible for regulating the real-time processing and the storage of the data is Storm. Storm is an event processor which consists of two components, spouts and bolts. The spouts are responsible for the input of data into the Storm topology. These input streams can then go to different bolts that perform processing on this data. After the processing, these bolts send the data to dedicated Kafka nodes responsible for the storage of the data.

6.1.4. Hadoop

The infrastructure of Metron is built around the Hadoop ecosystem in order to make it scalable for large amounts of data. Hadoop is an open source collection of software utilities that provides scalable and distributed computing by processing large data sets across clusters. This effectively allows for distributed computing and storage rather than relying on scaling the hardware itself vertically. Two technologies from the Hadoop ecosystem that are used in Apache Metron are Hadoop Distributes File System (HDFS) and Yet Another Resource Negotiator (YARN).

6.1.5. Pipeline

Apache Metron can be seen as a single large pipeline through which data is streamed. In this subsection this flow of data is described. The pipeline can be broken down into four parts: Telemetry ingress, parsing, enrichment and indexing of the data.

Telemetry Ingress

The first step of the pipeline is the telemetry ingress, here the network data enters the pipeline through the sensors described previously.

Parsing

In the parsing step, Metron parses the packets that come from each sensor. The data that comes out of the parser consists of packets which contain numerous fields. These fields

can be manipulated by changing the parsing configuration files in Metron that belong to the specific sensor. It is here where the requests to MaaS are configured. The parsing happens inside of the Storm bolts mentioned in subsection 6.1.3.

Enrichment

After the data is parsed, it ends up in the post-processing steps. In the enrichment period simple metrics can be performed on the data. For example you can derive a score by looking up in a blacklist whether certain fields of the packet are malicious. The score will then be added as a new field in the same packet.

Indexing

This is the final step where the packets are indexed and stored in HDFS and Elasticsearch. Once the data is stored it can be displayed through the web interface of Metron.

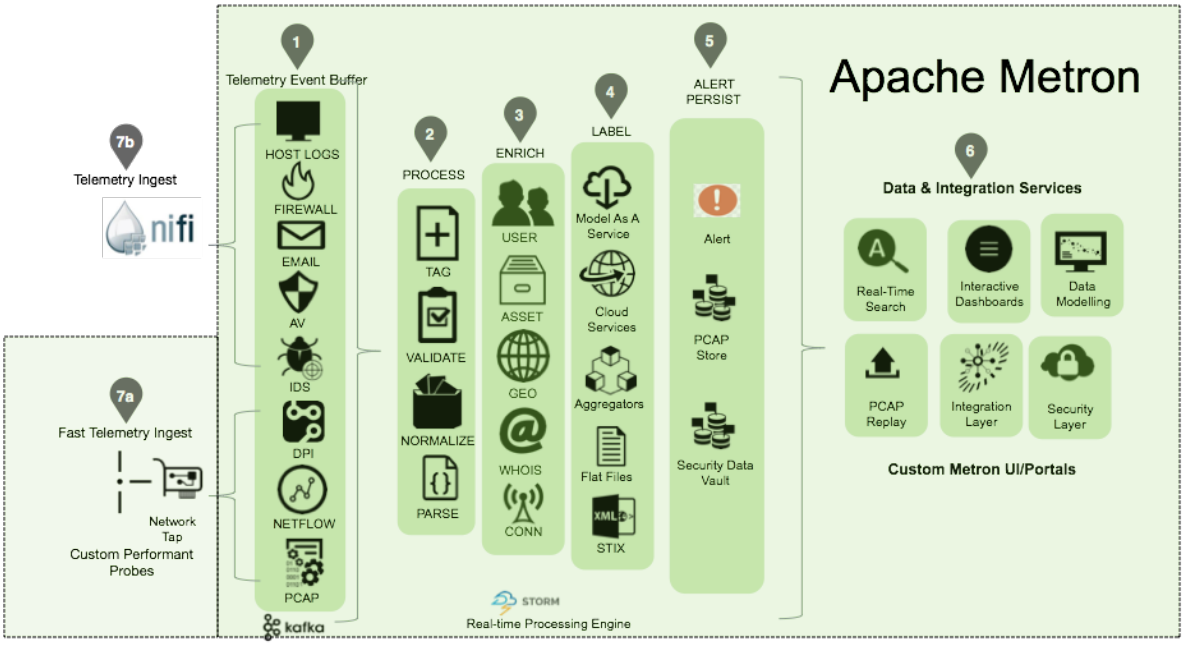

Figure 6.2: Metron architecture

6.1.6. MaaS

MaaS is the part of Apache Metron pipeline where custom models/code can be executed to enrich data. In the Apache Metron pipeline, a certain type of packet can be enriched by a specific MaaS, this is done by Metron making a request to the corresponding MaaS with part of the packet data such as IP address, hostname etc. The MaaS then does its analysis on that data and returns a value. This value is then appended to the original data packet in the Apache Metron pipeline.

Metron can only call the MaaS through HTTP requests, therefore in order to make the API calls the MaaS has to be a REST API. The deployment of the REST API is regulated in Apache Metron through Yarn.

REST API

The REST API works as follows: it listens to requests from Metron, these requests contain the input for the neural network. However these inputs may not be directly used by the neural network, as a neural network only accepts numbers, to understand why this is the case a short summary of neural networks is given in section 4.2. Therefore the inputs are first transformed using custom code that the user has to provide. When the input is transformed, it is fed into the neural network. The result from this neural network is thus a prediction on the input data. The REST API then sends back the response with the prediction that was made using the neural network.

Configuration
For configuration the only option is to use a configuration file. There can be no GUI as this program runs on a server so a configuration file is used to store all the settings for the REST API server. This configuration file contains the location of the model and which custom functions to use for each neural network input.

Custom code
The custom code is located in a single script. Here the functions are defined that are responsible for transforming the input data from Apache Metron to valid neural network inputs. Custom code was necessary to make this REST API server as versatile as possible. If there was no custom code functionality all the preprocessing would have to be done in Apache Metron which is very hard as the programming language used in the Apache Metron pipeline, *Stellar*, is very limited. Doing the preprocessing in Apache Metron would also lead to many extra unnecessary fields to be added to each packet which are then all stored in Elasticsearch for no good reason. Intermax stated that they did not want to bloat their Elasticsearch database just for the preprocessing of the data so custom code was a requirement for the REST API server.

YARN
YARN listens for model deployment requests and stores their endpoints in Zookeeper so that it can be identified by Apache Metron. Zookeeper is a queue that keeps track of the requests made by YARN and makes sure they are carried out in the right order. First it has to open a service that will listen for requests. Next the REST API will be deployed through a script where the API needs a unique name and version so that it can be identified. Additionally, the deployment requires a path to the local directory, which contains the files of REST API. The files in this directory will be copied and stored in HDFS so that Metron can execute it consistently and efficiently.

6.2. Neural Network Framework
The neural network framework is the part of the system where a user can create a neural network model, train it using input data and export it for use on the REST API server. The framework is CLI based. The user runs the CLI with a configuration file and the program then handles the training data input and training and exporting of the neural network model. This framework is intended to run locally on a users computer to have high development speed and easier tweaking of the neural network. It was decided that this framework is separate from the REST API, there is no direct communication between them. The reason for this is that Intermax indicated that transferring the model to the REST API shouldn't be automated as not to overload the server with deployments when simply testing out different models. Therefore the deployment of trained models will be done manually. A diagram of the Neural Network Framework can be seen in Figure 6.3, the usage of the neural network framework is described in section 8.1.

6.2.1. Command line interface
The program starts with a command line interace (CLI). This is simply a program that is run with the first CLI argument being the filepath to the config file. The flow of the program always returns to the CLI, meaning all calls to the machine learning library are made from the CLI. This was done to keep things as simple as possible and easy to debug.

The reason a CLI-based framework is used is because a graphical user interface (GUI) based tool would take too much time to develop in combination with the REST API server and the time lost to the research phase. A GUI was also not a top priority for Intermax as they were more interested in the application of the tool than the user friendliness.

6.2.2. Configuration
With a CLI-based tool configuration options are pretty minimal. One can either do it file based or via CLI arguments. The team decided that a file based configuration was easier to

use for the end user as small changes to the configuration only require small changes to the config file. A CLI argument based configuration would quickly become hard to read and way too long for a single command as one adds more settings.

The configuration file contains the settings for the neural network. Settings that are configurable are: the layout and size of the layers, the location of train and test input data files, where to export the model to and the amount of training steps. We did not make other settings tunable such as the types of training algorithms, this would be very hard to express in config files as there are many different optimizers. This means that users are limited to constructing neural networks that consist of a simple layout. This is however necessary as the complexity would become too high for a project of 10 weeks.

6.2.3. Neural network

The neural network part of this program consists mainly of a machine learning library and the binding between it and our CLI program. We decided to use a machine learning library instead of rolling our own neural network code as the point of the project is not to develop the best neural network algorithm but a framework around it. So while we may lose some customizability we gain a lot of development time that doesn't need to be spent on implementing a neural network on the lowest level. Therefore all the following steps regarding the neural network are done using a machine learning library.

The neural network as specified in the configuration file is first constructed. When the model is constructed it is trained using the train input data. After the model has been trained it is evaluated using the test input data resulting in the *real* accuracy of the model. It is important to always test your model on data it hasn't trained on, otherwise a high accuracy can mean the model is overfitting.[13] Finally, the model is exported to the local filesystem so it can be transferred to the REST API server where it can be applied on real data.

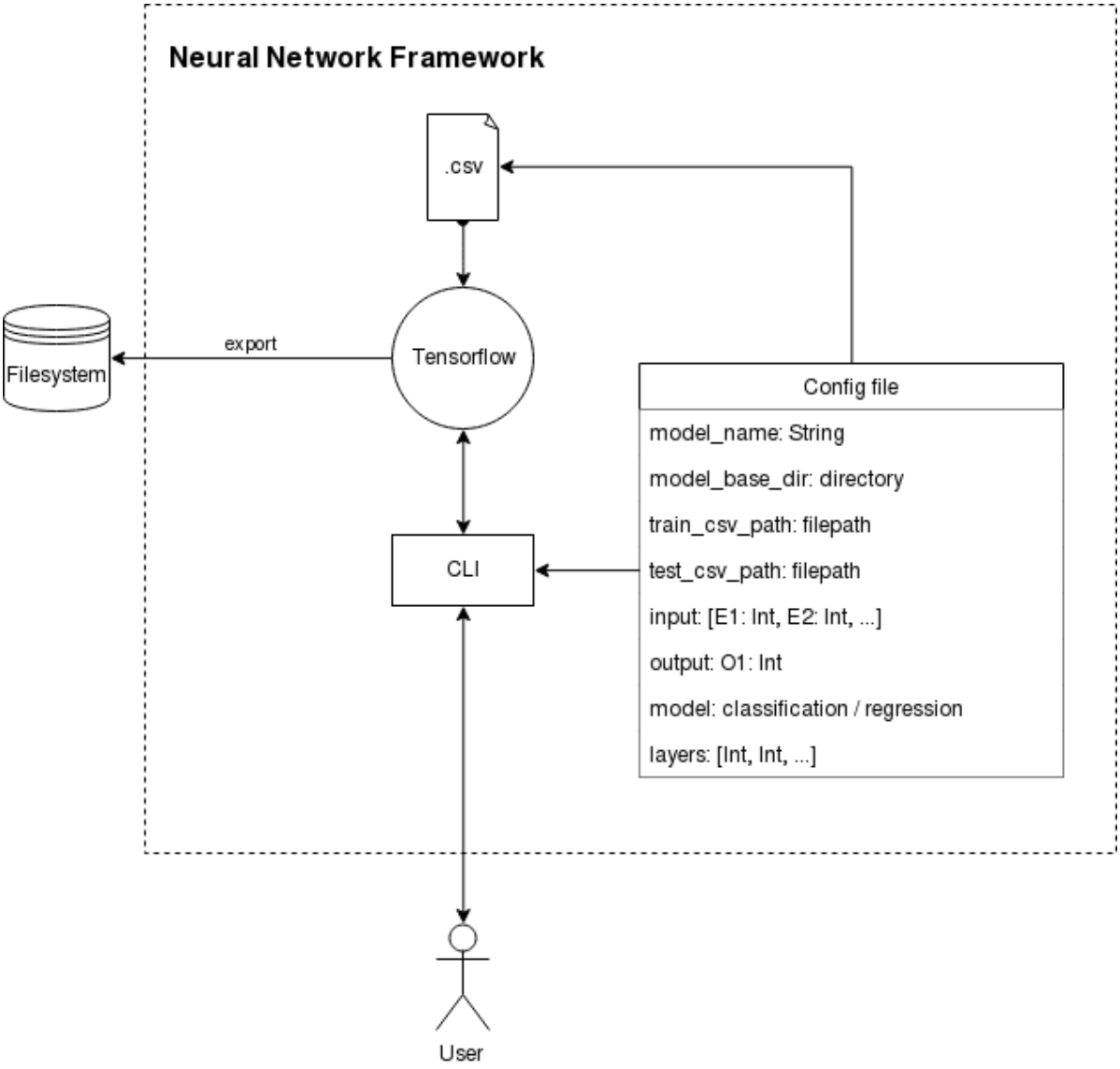

Figure 6.3: Neural Network Framework diagram

6.3. Workflow and data flow

In this section, the workflow and data flow will be described from two perspectives. One is from the user perspective, which is the person developing/deploying the neural network. The other point of view is the data perspective, as seen by a packet in the Apache Metron pipeline.

6.3.1. User perspective

The user of the system will be the one who will develop a model using the neural network framework and afterwards deploy the framework in Apache Metron as a MaaS. The steps of this process are as follows:

1. The user specifies the output location and characteristics for the model in the config file.
2. The user uses the neural network framework to construct an train a model.
3. The user transfers the model from the output location to the server where the MaaS is to be deployed.
4. The user configures and deploys the MaaS in Apache Metron.

5. The user configures Apache Metron to enrich packets using the MaaS.

A detailed manual of this workflow can be found in Appendix D

6.3.2. Data perspective

When a data packet arrives in Metron, it goes through a pipeline. At a certain point in the pipeline the data packet is send to the MaaS, which does some sort of data analysis on the data packet and returns the analysis back to the pipeline. Afterwards the analysis is used to determine a threat score on the package. The steps of the process are as follows:

1. The data packet enters the Metron pipeline
2. The data packet is parsed by Metron[1] to json format.
3. The data packet could be enriched in Metron[2] with additional information like a Geo location
4. The data packet is sent through a GET request in Metron with (part of) the packet as parameters to the MaaS REST API[3]
5. The data packet is transformed by MaaS to a neural network input using custom functions
6. The data packet is fed by MaaS into the neural network and transformed to contain the result
7. The data packet with the result is sent to Metron
8. The data packet with the result is appended to the packet in Metron

[1] https://metron.apache.org/current-book/metron-platform/metron-parsing/index.html
[2] https://metron.apache.org/current-book/metron-platform/metron-enrichment/index.html
[3] https://metron.apache.org/current-book/metron-analytics/metron-maas-service/index.html

7

Programming languages and Frameworks

In this chapter the decision making process behind the choice of programming language and frameworks is explained. The pros and cons of all choices are considered and weighed against each other to finally arrive at a set of tools the team can use to effectively and efficiently implement the system as laid out in the architecture design, chapter 6. First the choice of programming language will be discussed, subsequently the choice of frameworks will be made for the various parts of the system.

7.1. Programming languages

In this section the choice of programming language will be justified. The Bachelor End Project has a duration of 10 weeks total, of which 7 weeks are spent on programming due to the research phase and presentation preparation time. The team decided to rule out using programming languages that haven't been used extensively before by the majority of the team in order to save learning time. This restriction led to two remaining languages which would be suitable for the project: Java and Python.

7.1.1. Pros and Cons

	Pros	Cons
Java	- Team has a lot of experience with this language - Relatively fast compared to Python	- Slower development speed due to verboseness - Less industry standard frameworks for machine learning
Python	- High speed of development relative to Java - Lots of mature machine learning frameworks - Simple syntax enables easier custom code for the REST API that end users need to provide	- Team has a less experience in Python than Java - Might be slower than Java due to it being interpreted vs compiled.

Table 7.1: Pros and cons of different programming languages that could be used

In the end, Python turned out to have the upper hand in terms of pros and cons. The high speed of development and especially the custom code are big pros. On top of that, the speed of the programming language is not of utmost importance because Apache Metron will be able to scale multiple instances of the REST API. Machine learning frameworks in Python also optimise a lot under the hood in C so the actual speed difference is not as substantial as Java vs Python compared directly.

7.2. Frameworks

In this subsection the different framework choices are discussed. There are numerous frameworks for REST API's and Machine learning. Each framework has its own benefits and drawbacks. By considering the demands of the system that we designed, we chose the ones that met its requirements and were the most efficient to use.

7.2.1. REST API

For the REST API a web server is needed that efficiently handles many subsequent requests and has a robust REST interface. There will be only one HTTP endpoint to talk to so a fully fledged web server with a lot of dependencies is also not needed. With the programming language being Python several web servers frameworks come into the picture:

- **Flask**

  Flask is a popular framework used for web server development. It is a battle tested framework that is both easy to use while still being relatively performant. It also only focuses on the things we need which is simply being a web server.

- **Django**

  Django is a popular framework for web development and making REST servers, because it has many features. However due to all those extra features, the complexity of using this framework is higher than micro frameworks.

- **Falcon**

  Falcon is a fast web server framework focused on speed. This framework is faster in comparison to the other frameworks. But the speed of the web server is not the highest priority, because Metron can scale the REST API horizontally anyway. The class based API is a bit more convoluted than the style used by the other frameworks.

- **Bottle**

  Bottle focuses on both web server functionality and web page templating. This makes Bottle less appropriate than the other frameworks, because it focuses on forming web pages instead of answering simple HTTP requests.

From the comparison it appears Flask is the most fitting framework. Out of all frameworks we considered Flask was one of the most battle tested and light weight frameworks for simple HTTP requests which is exactly what we needed.

7.2.2. Machine learning framework

In the Neural Network Framework there is a need for a machine learning framework to train and apply neural networks. There are a few major frameworks that are used for machine learning in Python:

- **Tensorflow**

  Tensorflow is the state of the art framework for building neural networks that are ready for use in production. The framework is built by Google Brain, an AI research team at Google. It has great portability. A model trained on one machine can easily be deployed on another. The downside is that the framework is quite verbose and there is some boilerplate code. However this is expected for an enterprise focused machine learning framework and not necessarily a bad thing.

- **PyTorch**

  PyTorch is a machine learning framework developed at Facebook. It is arguably easier to use than Tensorflow because there is almost no boilerplate code. The disadvantage is that is is much less focussed on deploying models to production environments than Tensorflow is. It is therefore more suitable for research purposes than production ready models.

- **Scikit-learn**

  Scikit-learn is a kind of jack of all trades framework for machine learning, it has a wide variety of model types such as clustering, random forests, support vector machines etc. It also doesn't focus on production level deployment, it focuses much more on research instead.

From this comparison Tensorflow comes out as the best option. The library is focused on the saving and deployment of models in a production environment which is a major advantage over the other two frameworks which don't have this.

8

System usage

In this chapter the usage of the software product will be explained. The difference between this chapter and the architecture design in chapter 6 is that this chapter is more focused on describing the usage of the system. The architecture design on the other hand focuses more on how the system works and how it is laid out.

The first section of this chapter will be about the usage of the Neural Network Framework, the second section is about the configuration and usage of the REST API.

8.1. Neural Network Framework

The first component of the system is the neural network framework. The framework is build to easily train and export models in the Tensorflow framework (see subsection 7.2.2). The user only has to configure the configuration file and deliver the train and test data to run the system. In combination with the Rest API (see section 8.2) the framework makes it easy to create and deploy a neural network model without any Tensorflow experience. To use the system the minimal understanding of neural networks is required, but to get good results a basic understanding of neural networks is important, as the framework will not pre-process the data and select the best model for the user. In the next sections the the different components of the network will be explained.

8.1.1. Neural network

As previously mentioned the neural network framework is based around Google's Tensorflow framework[1]. Tensorflow makes it possible to create relatively easy to extremely complicated models. The framework makes use of the premade Deep Neural Network Classifier estimator[2]. This estimator expects an input function that imports the data and processes it to input features and output classes, a list of hidden layers and the number of output classes. Other more in depth configurations can be given but the framework doesn't uses these and will therefore not be mentioned. After the correct input is given, Tensorflow will take care of creating, training and evaluating the model. Tensorflow will even take care of optimising the training process.

8.1.2. Command Line Interface

The way to communicate with the framework is via a CLI. When the program is started is expects a configuration file with all the configuration for the framework to run (see subsection 8.1.4). After this the program will start and import the data, configure Tensorflow to train and evaluate the model and after that export the model to a given directory. In future development more CLI commands with be added, like the choice to export a model instead of always exporting and more getting more benchmark information about the framework and model.

[1] https://www.tensorflow.org/
[2] https://www.tensorflow.org/api_docs/python/tf/estimator/DNNClassifier

8.1.3. Data import

Before the network can be trained, training and test data have to be imported by the system.The format of the input data has to be CSV and the training and test data have to be in two different files. The data has to be pre-processed before it can be used by the network and all input values have to be integers or floats. Pre-processing via custom function, like in the REST API, could be added as a feature in future development work.

It is recommended to normalise the data before importing it, as it will probably give better results, while it is not necessary. The train data will be used for training and the test data for evaluating. The evaluating results will be given as a measure of performance. The user should split the data in such a way that there are no duplicates in the test and train set, as this could give a wrong image of the performance of the model. The model has the ability to remember outputs for certain inputs, also known as overfitting, and will perform well on duplicated data point. The model will therefore perform much better in the evaluation period than it will when it is deployed. So the user should prevent duplicate data points in the test and training sets.

8.1.4. Configuration

The main configuration is done in the configuration file of the Neural Network Framework. This JSON formatted file contains the different parameters that are needed to run the framework. The location of the configuration file is given to the Neural Network Framework as the first program argument. The config required the following fields in JSON format (For an example see: subsection E.1.1):

- `model_name`: The name of the model. The `model_name` is used as the name of the directory were the model is exported to in the `model_base_dir`
- `model_base_dir`: The export directory for all models. Different models are exported to sub directories named as `model_name` in the `model_base_dir` directory.
- `train_csv_path`: Path to the CSV file containing the train input and output data.
- `test_csv_path`: Path to the CSV file containing the test input and output data.
- `input`: The object that describes how to get the input features for the neural network from the train and test CSV files. The name of the elements will be used as the name of the feature of the neural network and have to correspond with the names of the elements in the REST API configuration file. The value of the elements is the index of the column in the csv file were the data of this feature can be found. Index starts from 0. This makes it possible to select only certain columns, in random order, from a data set and makes the pre-processing fase for the user easier.
- `output`: The object that describes how to get the output features for the neural network from the train and test CSV files. The name of the element will be used as the name of the output feature in the neural network. The value of the element is the index of the column in the csv file were the data of the output feature can be found. Index starts from 0.
- `model`: The kind of model that Tensorflow will use. Only classification is implemented at the moment because of the use case described in chapter 9 and because classifications is one of the main functionalities where neural networks are used for. For future development work other models, like regression, will be added.
- `layers`: The list of integers. Each integer represents an hidden layer with the amount of neurons in the layer. These layers will be used by the Tensorflow model.

8.1.5. Export

The model base directory is chosen by setting the `model_base_dir` in the config file. When a model is exported it is saved in the `model_name` directory. In Figure 8.1 an example export

directory structure is given. Tensorflow creates an timestamp folder of the model to distinguish different versions of the same model. When there are multiple timestamp folders for the REST API to choose from, the latest model will be chosen. In the example the `model_name` is `saved_model`. When a model is exported to the local file system, it can be moved to another file system to use for the deployment of the model with the REST API, as long as the directory structure stays the same.

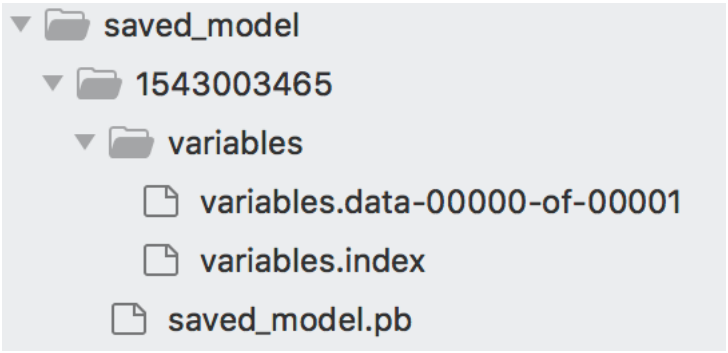

Figure 8.1: Example of export directory structure

8.2. REST API

The REST API's main responsibility is serving the model that was trained by the Neural Network Framework. It is necessary for scalability, having a single program running the model will not be enough for high throughput data. This is what brings the REST API into the picture, a REST API is primarily focused on handling concurrent requests from clients while making sure that requests are not held up by the processing of other requests. So while one request might take long another request won't notice this as the REST API server handles the other request in another thread. Furthermore REST APIs are able to be scaled horizontally as well, they are independent processes so if one REST API server doesn't provide enough throughput several others can be spun up to distribute the load.

The REST API is a general product, it can be used as a backend to any type of service. In the case of this project the main target was to implement a REST API server for the Apache Metron MaaS system.

8.2.1. Model

The REST API primarily serves the model. As described in subsection 6.1.6, the model is served by listening for requests in the `apply` route that contain the inputs for the neural network. These requests are simple HTTP requests with the data being inside the query parameters so all the data is confined into the URL string. An example request URL string could look like:

```
http://hostname:port/apply?qparam1=42&qparam2=41&qparam3=some_string
```

8.2.2. Configuration

The main configuration is done in the configuration file of the REST API. This JSON formatted file contains the different parameters that are needed to run the server. The location of the configuration file is given to the REST API as the first program argument. The config required the following fields in JSON format:

- `model_name`: The name of the model, this must be the same as the `model_name` given in the Neural Network Framework configuration, as this is the name of the folder the REST API will look for in the `model_base_dir`
- `model_base_dir`: The directory to look inside for the model called `model_name`
- `input`: The object that describes how to get the input for the neural network from the query parameters. Each element in the object corresponds to the input of the neural

network. The name of the element has to correspond to the name of the feature as given in the Neural Network Framework. The value of the element is a `function` object which has two elements:

- `name`: The name of the custom python function to call in the custom code file, this is further explained in the subsection after this
- `params`: An array of pairs, these are the parameters that are passed to the function identified by `name`. These parameters are passed to the function in the same order as they are defined in the array.

  A pair can look like `["key", "some_param"]` or `["var", "some_value]`. A pair with `key` means that the second value is the name of the query parameter that should be passed to the function. With a query parameter `qparam1=20`, `["key", "qparam1"]` would result in `20` being passed to the function identified by `name`.

  A pair with `var` is a static variable, this can either be a number, a boolean or a string, so `["var", 30]` would result in `30` being passed to the function identified by `name`.

An example of an input element is given below:

```
{
    "input_1": {
        "function": {
            "name": "to_feature_1",
            "params": [["key", "query_param_name_1"],["key",
              ↪ "query_param_name_2"],["var", 2],["var", "some_string"]]
        }
    }
}
```

An example of a complete and valid configuration file for the REST API can be found in subsection E.1.2

Custom functions

As mentioned in the previous section there is a `custom_code.py` file where function can be defined to transform the query parameters to valid neural network inputs (numbers). The names of these functions have to correspond to the values of the `name` field in the `function` object in the REST API config file.

An example function for normalizing a value:

```
def normalize(a, min_val, max_val):
    """
    Normalization function maps a value to a value between 0 and 1
    :param a: Value to normalize
    :param min_val: Min value
    :param max_val: Max value
    :return: The normalized value
    """
    return (a - min_val) / (max_val - min_val)
```

So having a REST API config with a single neural network input:

```
{
    "input_1": {
        "function": {
            "name": "normalize",
            "params": [["key", "query_param_1"],["var", 1],["var", 10]]
        }
    }
}
```

Would make the REST API look for a query parameter called `query_param_1` in each request and pass the value belonging to that parameter to the `normalize` function as the first argument. `1` and `10` are passed as second and third arguments to `normalize` respectively because of their position in the array. The result of `normalize` is then passed to the neural network. So if the following HTTP request is sent to the server:

`http://hostname:port/apply?query_param_1=2`

The result of the `normalize` function would be `0.125` The neural network then takes this input and returns a prediction. The prediction is then sent back as a response to the initial request that was made. This example only has one feature input but for real world models more feature inputs are used for the neural network. An example custom code file for a real application can be found in subsection E.1.2.

8.2.3. Deployment

The deployment of the REST API varies by application, if it is simply used as a backend server for a website, one can just run the startup script with the configuration file directly. However if the REST API has to be deployed to Apache Metron as a MaaS a different script should be used that has already been written by the team. This script is called `maas_deploy.sh` which handles the copying and the launching of REST API and registering it with Apache Metron.

9 Proof of concept

In this chapter the use of the neural network framework for finding malicious SSL/TLS certificates is investigated. In the last decade the use of SSL/TLS on phishing and command and control domains has increased exponentially [10]. Since browsers added the green padlock for domains that use HTTPS for communication, phishing websites started to use SSL/TLS to make their website seem more legitimate [10]. The price of certificates dropped in the last few years, making it profitable for hackers to buy large amounts of certificates for their phishing domains. On the other side command and control domains are using SSL/TLS to encrypt their communication with the infected hosts [25]. Data encryption makes it impossible for an IDS to detect malware and malicious activity through payload analysis. Therefore it becomes harder for researchers and cyber security agencies to investigate command and control schemes like botnets. Intermax is interested in detecting malicious certificates to detect infiltrated systems and data exfiltration. The possibility to detect phishing domains through phishing certificates will also be investigated, while this is a lower priority. As mentioned in the research report (Appendix A) different features will be extracted from the certificates. With these features a neural network classifier will be trained to classify malicious and non-malicious certificates. Based on the certainty of the prediction a threat score will be generated. This chapter will describe the selection of the right data, features and finally model. Afterwards the results of the different models will be discussed.

9.1. Data selection

The neural network that is used needs labelled data to be trained on. There are no publicly available SSL/TLS data sets. To train the model a data set of phishing, malicious and legitimate certificates had to be created. The legitimate certificates were gathered from scraping the first 300.000 domains on the Alexa top million domains[1]. The phishing certificates were collected by scraping the domains from phishtank[2]. The malicious domains were collected through matching the SSL sha-1 fingerprint on abuse.ch[3] with the CIRCL Passive SSL API[4]. Though the use of custom scrapers the team managed to collect:

Certificates	Alexa	Phishing	Malicious
Total Collected	111.007	11.744	908
Unique	68.655	2677	868

Table 9.1: Number of certificates collected

[1] http://s3.amazonaws.com/alexa-static/top-1m.csv.zip
[2] http://data.phishtank.com/data/online-valid.csv
[3] https://sslbl.abuse.ch/blacklist/sslblacklist.csv
[4] https://www.circl.lu/services/passive-ssl/

9.1.1. Uniqueness

For all certificates the duplicate certificates were removed. This is important to balance the data set which can prevent over fitting. When a neural networks is over fitting it is remembering the output of certain data points instead of learning general relationships between the input and output. When the training data set has a lot of duplicates it will just remember the outputs for these inputs. For the phishing certificates duplicates where a big problem. Only 23% of the phishing certificates was unique. The sha-1 fingerprint of the models were used to determine uniqueness. In section 9.4 only the unique certificates were used to construct the training and test data sets.

9.1.2. Data set creation

In order to train the neural network training data is needed. However one cannot just collect as much data as possible and throw it into the neural network expecting good results. There are two aspects that are important when creating data sets.

One aspect of data set creating is balancing. Balancing a data set is extremely important as an unbalanced data set can cause a model to have a hundred percent false positive rate. For example if the Alexa/Malicious data set would consist of 68.655 normal certificates and 868 malicious certificates (1.2% of certificates is malicious) the model would probably predict that all certificates are non-malicious, as it can get an 98.8% accuracy rate. This model would have a hundred percent false positive rate and would never predict that a certificate is malicious. In other words the model would be useless. Therefore the data sets were balanced to prevent this from happening.

Another aspect of creating datasets is the train/test split. This split of the data set separates the data set into a train set, on which the neural network is trained, and a test set, on which the performance of the neural network is evaluated. This is necessary to detect overfitting, which is when a neural network memorises the training data itself instead of recognising patterns in it.

Two different data sets have been created, an Alexa/Phishing data set and an Alexa/-Malicious data set. The number of unique phishing and malicious certificates is way less than the number of unique Alexa certificates. For the purpose of balancing the data sets an random selection of 2500 unique Alexa certificates was selected. In the Alexa/Malicious data set 25.7% of the certificates was malicious and in the Alexa/Phishing data set 51.7% of the certificates were phishing certificates. After the balancing the data sets were split into training and test sets. The ratio between training and test was 85% 15% for both datasets. The high training percentage was chosen because of the small amount of data points in the balanced sets, so that as much data points are used to train on. The choice to still keep 15% as test data was made so that there were still small performance differences visible in the different models. When a test set has a small amount of data points, one wrong classification can lead to a big accuracy drop. That could cause models to be wrongfully judged.

9.2. Feature Selection

Good features (input variables for the neural network) are essential for the performance of the neural network. When a model is trained with a lot of irrelevant features the relevant features have a low impact on the classification. Therefore the network will not work properly and have a high amount of false positives and false negatives. Good features are features that have a low correlation to each other because each feature with add new information about a data point relative to the other features. On the other hand too few features can prevent a neural network from learning because of the lack of information about the data point. In this section the process of selecting the right features is described.

9.2.1. Normalisation

Normalisation is the process of mapping the input data to a common scale. It is very common for neural networks to have a simple linear transformation normalising the data to a range

of 0 to 1 [22]. The formula for performing this transformation is as follows:

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Where:

X = The original value to normalize
X_{min} = Minimal value of X
X_{max} = Maximal value of X
X_{norm} = The normalized value of X

Note that this particular function transforms the input value to a range of 0 to 1. Normalisation makes it easier for a neural network to converge to certain weights. With non-normalised features some features may have an domain between 10 and 10 million while others have domain between 0 and 1. With normalisation each feature has the same domain and start therefore with the same impact on the end classification.

9.2.2. 43 features

The first models were based on 43 features extracted from the certificates (see Table G.2). After investigating the data set it was clear that a lot of the features were redundant and were not adding extra information to the data points. Another data set was created with less features.

9.2.3. 7 and 9 features

The next models were based on the 6 features with the highest information value in the first models and a Shannon entropy on the subject common name because of a noticeable difference between the malicious and non-malicious subject CN . For the last models two additional features were added. The first feature gives information about the self signing of certificates, which was more common in malicious certificates than in non-malicious certificates. The second features gives information about the life time of the certificates. A difference was noticed between the life time of malicious and non-malicious certificates. See Feature Table G.3.

9.3. Neural Network structure

The general structure of a simple neural network is that it starts with the input features that are fully connected to the first layer of neurons. Fully connected means that each node from one layer is connected to a node in the next layer. Then there can be multiple layers after this which are also fully connected. Finally each neuron in the last layer is connected to a single neuron for binary classification/regression. In the case of binary classification the output of that last neuron is the probability of the input being labelled as one of the two classes. In the case of regression the output of the last neuron is a value that has the same range as that of the training data.

The structure of the layers often looks like a cone, it starts very wide but it narrows down to a small number of neurons at the end. The reason for this is that neural networks are designed to concentrate information from a lot of features down into a single value. Going from a layer with one neuron to a layer with ten neurons is pointless as no new information can be gained from that single neuron. Therefore it is often a good idea to decrease the layer size as the network grows deeper.

9.4. Model Selection

Now that several models have been described with different amounts of features and normalisation it is time to empirically decide which model is better. The selection will be done according to several metrics described in the next section.

9.4.1. Metrics

There are different metrics used for evaluating Neural Networks and classification algorithms in general. Simple metrics are accuracy (ACC), true positive rate (TPR) and false positive rate (FPR):

$$ACC = \frac{\sum real\ positive}{\sum total\ population}$$

$$TPR = \frac{\sum true\ positive}{\sum real\ positive}$$

$$FPR = \frac{\sum false\ positive}{\sum real\ negative}$$

However these metrics don't deal with class imbalance. As explained in subsection 9.1.2 class imbalance is when there is a difference between the relative size between training data classes. Using TPR and FPR in a ROC-curve using an imbalanced dataset cannot properly distinguish well performing models from bad ones regarding false positives [24].

Since the Alexa/Malicious data set is still relatively imbalanced due to the lack of C2 certificates relative to good certificates from the Alexa top domains, different metrics have to be used than the ones mentioned previously. Two metrics called precision and recall are able to distinguish good from bad models that are using imbalanced datasets [24]. To compare the results from phishing and c2 models precision and recall will also be used for the phishing models. These metrics are defined as follows:

$$precision = \frac{\sum true\ positive}{\sum predicted\ positive}$$

$$recall = \frac{\sum true\ positive}{\sum real\ positive}$$

The precision calculates the ratio between the true positives relative and the true and false positive combined. Therefore a higher precision indicates that there are less false positives relative to the true positives, meaning that the model is quite conservative in classifying positive. The recall calculates the ratio between the true positives and true positives plus false negatives. Therefore a higher recall implies that there are less false negatives relative to the true positives, meaning that the model is generous in classifying positive.

For the use case of detecting SSL certificates used in C2 communication the precision was deemed more important than the recall. The reason for this preference boils down to the application of the classification. In Apache Metron alerts can be generated when it identifies a certificate used for C2. If the recall is higher than the precision of the model there will be relatively more false positives than false negatives. So while the model may detect almost every actual certificate used for C2 as a certificate used for C2, there will be a lot of false positives. Both the team and Intermax found that having too many false positives is not suitable as this would generate too many alerts while there is no real need for action in the first place. Therefore precision is more important than recall in this proof of concept. On the other hand the precision should also not be maximised as it is possible for a model to be so lax that is misses out on almost all of the C2 certificates, making the precision almost equal to one while the recall plummets to zero.

9.4.2. Results

	Phishing 43 features			Command and Control 43 features		
Layers	Accuracy	Precision	Recall	Accuracy	Precision	Recall
10,10	0.7036	0.7503	0.5827	0.9269	0.9098	0.8043
14, 10, 7	0.689	0.7574	0.5277	0.9269	0.9166	0.7971
25,20,15,10	0.7168	0.806	0.5478	0.9192	0.9	0.7826

Table 9.2: Neural Network performance comparison for the phishing and command and control certificates on 43 features

Phishing	7 Features Normalized			9 Features			9 Features Normalized		
Layers	Accuracy	Precision	Recall	Accuracy	Precision	Recall	Accuracy	Precision	Recall
10,10	0.7107	0.793	0.6384	0.6889	0.7857	0.5938	0.6709	0.7972	0.5352
14, 10, 7	0.6966	0.7878	0.6103	0.6902	0.7811	0.6032	0.6812	0.7392	0.6455
25,20,15,10	0.7082	0.7884	0.6384	0.6992	0.7758	0.6338	0.6876	0.7715	0.6103

Table 9.3: Neural Network performance comparison for the phishing certificates on 7 normalized, 9 and 9 normalized features

C2	7 Features Normalised			9 Features			9 Features Normalised		
Layers	Accuracy	Precision	Recall	Accuracy	Precision	Recall	Accuracy	Precision	Recall
10, 10	0.9131	0.9196	0.7744	0.9198	0.9369	0.7819	0.9175	0.9363	0.7744
15, 15	0.9153	0.9279	0.7744	0.9223	0.9375	0.7894	0.9287	0.9316	0.8195
20, 20	0.9131	0.9196	0.7744	0.9223	0.9454	0.7819	0.9242	0.9459	0.7894
8, 8	0.9064	0.8888	0.7819	0.902	0.8739	0.7819	0.9086	0.9339	0.7443
5, 5	0.9042	0.8813	0.7819	0.9042	0.8813	0.7819	0.9198	0.9369	0.7819
8,6,4	0.9198	0.9145	0.8045	0.9086	0.8965	0.7819	0.9109	0.9266	0.7593
10,7,5	0.9131	0.8852	0.812	0.9064	0.8823	0.7894	0.922	0.9152	0.812
14, 10, 7	0.922	0.9298	0.7969	0.9153	0.913	0.7894	0.9086	0.9509	0.7293
7,5,3	0.9064	0.8823	0.7894	0.9198	0.9449	0.7744	0.9153	0.9279	0.7744
20, 15, 10, 5	0.9131	0.9051	0.7894	0.9175	0.921	0.7894	0.9354	0.9333	0.8421
25,20,15,10	0.9242	0.9459	0.7894	0.9242	0.9459	0.7894	0.9420	0.9495	0.8496

Table 9.4: Neural Network performance comparison, note that the model highlighted in green is the chosen model for this proof of concept

At first, the model of 43 features was measured. The first models were trained on phishing certificates and the others on C2 certificates. The results of these measurements are shown in Table 9.2. The results for command and control are significantly better than the results for phishing. After the 43 feature model the team trained the 7 and 9 feature models, again both phishing and C2 certificates were used as training data separately. The results for the phishing certificates can be seen in Table 9.3 and the results for C2 certificates are displayed in Table 9.4. From these three tables it becomes apparent that in general phishing certificates perform worse than C2 certificates in these neural networks. For phishing the highest results were an accuracy of 72%, precision of 81% and a recall of 65%, these are not practically useable as 1 in 5 classifications are false positives and $\frac{1}{3}$ of the phishing certificates are classified as benign. Furthermore these scores are the best across all models so they are not even achievable using a single model.

The C2 certificates however, do show promising results. From Table 9.4 the best model regarding features and layers has 9 features and the layer structure of the model is $[25, 20, 15, 10]$. The reason that this specific model is the best is because of the high precision and recall of 94% and 83% respectively. Only the $[14, 10, 7]$ layered model improves upon the chosen model regarding the precision, however the recall of those models is much worse. Recall is still important as it is also undesirable to have a decrease of recall by 12 percent points only for the precision to increase by only 0.1 percent point.

Regarding the normalised versus non normalised, Table 9.4 shows that in general the performance for normalised data is higher than non normalised data. Both the precision and recall are often higher for normalised models than the non normalised variant. Therefore the final model chosen is of the normalised variant.

After analysing the different models regarding layers, features and normalisation the final model that will be used is the $[25, 20, 15, 10]$ model with 9 features that are normalised. This model is highlighted in green in Table 9.4.

Precision and recall vary on the threshold that is used to classify items. This threshold determines when a certain item is classified as positive or negative. For example if an item is classified as label A with probability 70% and label B 30%, and the threshold for classifying A is 50% the item will be classified as A. However if the threshold for classifying A would have been 80% the item would be classified as B since the probability wasn't high enough. This balancing act of selecting the threshold is a matter of preference for precision versus recall. This balancing act can be visualised in a precision recall curve. This curve plots the

precision against the recall. Each point on the curve represents a probability threshold for which the precision and recall are the y and x coordinates respectively. In Figure 9.1 the precision recall curve is depicted for our chosen model. Note that for tables 9.2, 9.3 and 9.4 a probability threshold of 50% is used.

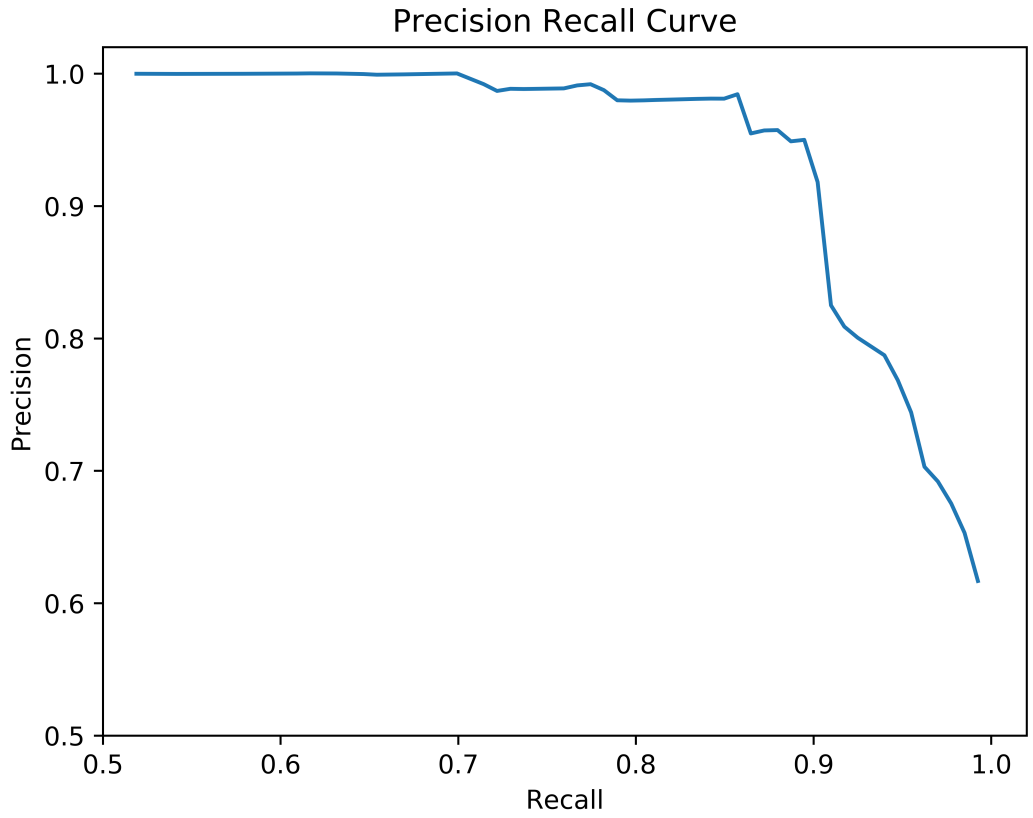

Figure 9.1: Precision recall curve for the $[25, 20, 15, 10]$ model with 9 normalised features

This proof of concept will only send the probability itself back to Metron as a threat score, it is not necessary to do classification because Metron can do more advanced bucketization with threat scores than simple binary classifications. Since this proof of concept only needs to send the probability of the certificate being a C2 certificate to Metron there is no need for a probability threshold.

9.5. Conclusion

By researching several types of models and empirically testing these out, the team decided which one would be used. From the results it became apparent that the phishing certificates proved to be more difficult to get good results with than C2 certificates. However this was not a big problem as Intermax's initial goal was to detect C2 certificates in the first place. The final model that the team used has a relatively good accuracy of 94%, precision of 94% and a recall of 83%.

9.6. Discussion

The phishing certificates turned out to be much harder to train the models on than C2 certificates. One of the reasons for this is that phishing certificates need to be as real as possible, as to not tip off the browser of the victim. For C2 certificates however this is not the case at all. Most of the time C2 communication only cares about the encryption leveraged by these

SSL certificates, therefore they often just use self signed certificates because it is easy to generate and use, thus making them easier to detect compared to phishing certificates.

The other problem we encountered was that of root CA certificates. Since these certificates are self signed they are detected as C2 certificates. Since neural networks train to generalize data it is hard for them to train these certificates into the model itself as a whitelist. It might even decrease the accuracy of the model as it has to add new constraints into the network which may cause other connections in the network to be less effective than before. Therefore we don't train the neural network to recognize root CA certificates, instead this can be done much more efficiently in Apache Metron before the REST API is even called to save time.

10

Validation and Verification

10.1. Validation

In this section the system will be validated. Validation is the act of verifying whether the system actually solves the problem of the stakeholders according to the requirements.

Requirement	Satisfied
The framework must be able to generate a model from a given data set	✓
The framework must be able to give accurate predictions based on the input data	✓
The framework must be able to provide a classification model	✓
The framework must be able to export the generated model	✓
The user must be able to specify the input data and neural network configuration through a config file	✓
The user must be able to start the framework through a command line interface (CLI)	✓
The user must be able to specify the config file location for the framework	✓
The API must be able to import the generated model	✓
The API must be deployable by Metron	✓
The API must be able to return an error message if incorrect input data is passed through	✓
The API must return a classification of the input data	✓
The API must be deployed in such a way that	✓
The API must run consistently with Metron without crashing	✓
An API instance must be independent of other API instances	✓
The user must be able to specify the model location through a config file	✓
The user must be able to specify the config file location for the REST API	✓

Table 10.1: Must have requirements satisfaction

Requirement	Satisfied
The classification should include an accuracy score	✓
The API should not delay the data stream significantly	✓
The API should be able to transform input data to neural network features using custom code	✓
The user should be able to verify the model using specified test data	✓
The user should be able to see when the training of the model is done	✓
The user should be able to decide whether the model should be exported or not	✗
The user should be able to specify the input transform function for the API through a config file	✓

Table 10.2: Should have requirements satisfaction

10.1.1. Functional Requirements satisfaction

A simple way to see if the system that was developed solves the problem is by taking a look whether the requirements of the stakeholders are satisfied.

In Table 10.1 the Must have requirements are depicted along with a checkmark indicating they are satisfied. All Must haves of the initial requirement specification have been implemented and therefore satisfied.

Requirement	Satisfied
The framework could output benchmarks	✓
The user could be able to specify whether error message are returned	✗
The user could be able to see a progress bar of training the model	✗
The user could be able to specify a port number in the config file	✗

Table 10.3: Could have requirements satisfaction

Table 10.2 shows the satisfaction of the Should have requirements. A cross indicates the requirement has not been satisfied. As can be seen, not all requirements have been satisfied but this is not a major setback. According to the MoSCoW method, Should haves are requirements that are still important but not critical to the project [1]. Therefore not having two out of seven Should have requirements is not detrimental to the functioning of the system.

The Could haves are displayed in Table 10.3. From this Table, only one out of four requirements has been satisfied. This is to be expected as Could haves are features which are nice to have, but are only implemented if there is enough time to implement them [1].

Since Won't haves are not meant to be implemented, the team didn't any of them [1]. Since the Won't haves aren't implemented they can't affect the system in any way.

10.1.2. Use case: C2 communication SSL certificate classification

The use case of detecting SSL certificates used in C2 communication has been solved by using the system designed and developed by the team. In chapter 9 the proof of concept validates that the system solves the problem as stated by Intermax.

On top of that, throughout the project there have been weekly meetings with the clients. These meetings were primarily used to validate the progress of the product. The meetings in the first few weeks were used to design a product and in the last weeks were focused on the implementation and validation part. The conclusion of these meetings was that the developed system did solve Intermax's problem effectively, therefore the system is also validated by the stakeholders. The stakeholders are the most important party involved when validating the system, because they are the customer. Therefore having the stakeholder validate the system is a major achievement in terms of system validation.

10.1.3. Neural Network Framework usability

The tool for creating neural networks was validated by comparing it to directly using Tensorflow manually. Using the CLI in combination with the configuration file resulting in a much higher speed of development as opposed to using Tensorflow directly. Using the CLI one was able to just change a couple of variables in the configuration file to tweak the neural network or provide other train/test data. When using Tensorflow directly these different variables have to be changed in different locations of the code. Another advantage of the framework is that it is easier to keep different versions of a model configuration on a file basis, whereas when using Tensorflow directly one would need to save revisions of the whole codebase or manually copy paste the variables into the code every time when trying a new configuration. Therefore the improved usability of the CLI over manually using Tensorflow validates the Neural Network Framework.

The requirement satisfaction, use case, and usability have proven the right system has been built for the solution, because the system solved the problem efficiently and successfully. The system was also just able to be built in the given timeframe indicating that a more sophisticated system with perhaps a full blown GUI would have resulted in an unfinished product. Most importantly the stakeholders are satisfied with the product as it solves their initial problem statement. As a result the system has been successfully validated.

10.2. Verification

In this section the system will be verified. Verification is the act of checking whether the system actually works and if it was implemented the right way. First the functionality of the system is verified after which the implementation is verified.

10.2.1. Neural Network Framework Verification

The neural network framework can be verified by using the functionality that it provides and checking whether it behaves as expected. This can easily be checked through manual tests. Several models have been trained using the neural network framework. In chapter 9 several models were constructed. These models thus verified that the neural network framework successfully constructed different models.

10.2.2. REST API verification

The REST API is similarly verified by actually sending HTTP requests to the server and expecting back a correct response. When invalid input was sent to the server it correctly returned an error message. When a correct input request, crafted according to the configuration file, was sent to the server a valid score was returned. This verified the behaviour of the REST API server that it shouldn't crash upon unexpected input and correctly send back a score when valid input is given.

10.2.3. System verification

To test the system as a whole the proof of concept of SSL certificate classification is applied and implemented. First the model is trained using the Neural Network Framework. This model is then transferred to the Metron server. On this server the REST API is started using the deploy script. The user then makes network requests on a machine that is monitored by Apache Metron to get network data into the Metron pipeline.

After connecting to several C2 domains obtained from a list of C2 domains[1] we found that our REST API successfully classifies them as certificates used for C2 communication. Out of all domains that were connected to, Metron classified them all as C2 communication certificates. The results ranged from 97.85% to 99.93% probability of the certificates being used for C2 communication. Therefore we know the REST API classifies the C2 certificates correctly using the trained model.

Knowing that the model works on C2 certificate domains the team tried to connect to benign domains obtained from the Alexa top domains[2]. These sites were also all correctly identified as benign certificates. The results of these classifications ranged from 98.68% to 99.97% probability of being benign. Therefore we can safely say that the certificates used for normal traffic are correctly classified by the REST API as well, indicating correct behaviour of the system.

As can be seen from the results all parts of the system function correctly from initial training to deployment. Furthermore the interaction between the components also works as our REST API correctly interfaces with Apache Metron. Therefore the workings of our system are verified by this use case.

10.2.4. Scalability and Speed

One of the requirements is that the system shouldn't delay the Metron pipeline significantly. This means that the REST API should be quick to respond to requests. The delay of the REST API server was measured in a best case scenario, namely sending requests to a locally hosted server. The time it takes to get a response from the server is measured in milliseconds. The results are displayed in Table 10.4. From this table it becomes clear that the REST API only imposes about a millisecond delay at best. Only the network could impose a further delay but this is not something that can be influenced by the REST API. Therefore the speed requirement has been verified by this benchmark.

[1] http://osint.bambenekconsulting.com/feeds/c2-dommasterlist.txt
[2] https://www.alexa.com/topsites

Because of the high speed the scalability also improves significantly. As the REST API instances are independent of each other, multiple instances can be run in parallel. This however would not make any difference if requests would take for example a second. If a request would take one second it doesn't matter how many REST API instances one runs in parallel, the throughput will stay at 1 request per second per server, so to serve 5000 requests per second 5000 REST API instances are needed. However, with a response time of only 1 millisecond a throughput of 1000 requests per server is achieved, so one only needs 5 REST API instances to serve the same 5000 requests per second. Therefore the high speed of the REST API server also enables it to be scalable.

Average (ms)	**Median** (ms)	**Standard deviation** (ms)
1.000547409	0.997066498	0.010721522

Table 10.4: Table showing the average, median and standard deviation of the request time to the REST API server in milliseconds, see Table J.1 for original data

10.2.5. Non-functional requirements satisfaction

The non-functional requirements are the requirements that focus on the operation of the system rather than the behaviour. The satisfaction of the non-functional requirements is shown in Table Table 10.5. This table shows that all non-functional requirements from the problem analysis are satisfied. Therefore the satisfaction of the non-functional requirements verify that the system operates correctly according to these requirements.

Requirement	Satisfied
The system must be incorporated into Apache Metron as a REST API	✓
The API should be able to run on Linux	✓
There should be a user manual for the framework	✓
There should be a user manual for the API	✓
The testable code should have a branch coverage of >75%	✓

Table 10.5: Non-functional requirements satisfaction

10.2.6. Testing

Testing is important in any type of software project. Without tests it is impossible to know if all the code still works after changing small parts of it, making it very hard to maintain the codebase. By writing tests one can immediately see when a change in certain parts of the system breaks behaviour in another, of course the tests have to be written in the first place to see this behaviour. The team used a testing framework for Python called *pytest*. This framework was used for almost all of the tests. Besides that library, the built-in Python functionality for mocking objects was used, namely *Magicmock*.

Unit Tests

Unit tests were written on a function basis. This means that a function is tested on its own, while mocking away objects that it uses so only the functionality of that function is tested. Therefore unit tests verify the codebase on a function level.

Integration Tests

Integration tests test the behaviour of multiple function or classes interacting with each other. These often only mock system dependencies and external libraries that are used. Therefore integration tests verify the behaviour of parts of the codebase as a whole.

Manual Tests

Finally the manual tests test the behaviour of the system as a whole. In this case nothing is mocked away so the interaction between the filesystem and external libraries are verified

this way.

The unit and integration together cover 82% of the codebase based on branch coverage. This exceeds the coverage of the initial requirement stating a coverage of 75%. The manual tests don't contribute to the coverage as it would jump up to nearly 100% immediately because it executes all of the code, including the calls to libraries, the system etc.

10.2.7. Software Improvement Group Code Quality Assessment

The TU Delft wants their students' code to be evaluated by the Software Improvement Group (SIG)[3]. The SIG assesses the code based on multiple metrics which have not been communicated to the team. The only available metrics to the group, are the metrics on which the SIG gives feedback. Next to that, a score is given based on the maintainability model of the SIG in the form of a number of stars. The SIG did, however, not communicate the minimum and maximum of the amount of stars that can be received.

The REST API and the Neural Network Framework were uploaded as two separate repositories. These repositories were uploaded at the end of week 6 and feedback was received in week 8. The REST API scored 4.1 stars and the Neural Network Framework scored 3.8 stars. The exact feedback of the SIG can be found in Appendix I. The feedback was processed and in week 9, both repositories were uploaded again to the SIG. This feedback has not been received yet at the time of writing and is expected just before the presentation.

[3] https://www.softwareimprovementgroup.com/

11

Ethical considerations

While often overlooked, ethical considerations always need to be made when developing a product that involves data generated by humans. In the case of this project these ethical considerations were made from the point of view of Intermax, as this is the only place where the product will be used. In the following paragraphs the ethical consequences of the system will be given and the way Intermax manages these is explained.

First of the product itself does not collect any user data, the data merely passes through the system. This makes it impossible for the system to be responsible for the information that goes through it, the only data it is responsible for is the output data. The output data is determined by the neural network model which in turn is configured by the end user (employees at Intermax). The only way for unethical practices to take place is when an end user constructs a model which is not ethically responsible. So indirectly the system developed by the team can facilitate the use of potential unethical models by end users. Therefore the system itself does not have ethical issues, however the use of it does.

Another point is that Intermax has several certifications it needs to conform to.[11] Notable certifications are ISO 27001 which is about information security [12], and NEN 7510, which is the Dutch standard for security information in the Dutch healthcare sector [21]. These certifications require Intermax to conform to high privacy standards regarding privacy sensitive data, especially electronic patient information. Therefore the system developed by the team also has to conform to these standards when it is eventually used in production. As a result Intermax will have to use the system responsibly as third party certifiers will audit this system. These certifications will therefore enforce the proper usage of the developed system regarding information security, privacy of electronic patient information and other user data.

12 Discussion

In this chapter the various expected and unexpected challenges that the team faced are discussed and the solutions the team came up with are clarified. Followed by the improvements that can be made for future projects.

12.1. Tensorflow

One of the issues the team faced was the continuous integration. Tensorflow has focused so much on performance improvements that it requires the AVX CPU instructions[1], which significantly improve calculation speeds. Unfortunately the CI servers of Intermax were virtualized and didn't have this setting enabled and it wasn't possible to change this either. The solution to this problem was to run the test stage of the CI script locally and copy/paste the resulting test report as a comment in each pull request.

12.2. Metron

Apache Metron, while being a feature rich and actively developed system, is not mature as of yet. In the following paragraphs the issues the team faced and how they were solved will be discussed.

In order to use the MaaS of Metron there were prerequisites, some knowledge in the area of Linux and Apache Hadoop is needed. In general learning these prerequisites are not troublesome. The real problem here is finding out whether it is done correctly or wrong. Metron, MaaS in particular, has a lack of error logging and the ones that exists didn't show much. For example, the script that was used to deploy the REST-API required many variables to be deployed. When one of these variables were missing, the output of the script was the same as when the variables were included. Making mistakes when using something new is quite natural, but having no clue where or what these mistakes are makes it difficult and time-consuming to solve them.

Metron itself is not so well documented. Editing the configuration files of the parser required a restart of the parser services in the Ambari management interface instead of the Metron interface, which wasn't documented. This was time-consuming to find out and use, because restarting took about 10-20 minutes for each iteration. The lack of error logging in Metron effectively turned this method into a trial-and-error approach, without the error part.

Another problem relating to Metron was the folder structure collapsing on deployment. A current bug in Metron is that deploying a model collapses all the files in the subdirectories of the model into the root directory of the resulting container it is deployed in, essentially flattening the whole folder structure. This is not necessarily problematic for the execution of the source code as this can be refactored into a single folder, but it is problematic for Tensorflow. Tensorflow required the saved model to be in a certain folder structure in order to load it, this structure is described in subsection 8.1.5. In order to solve this a custom

[1] https://www.tensorflow.org/install/source

script was written to move the files into the same structure as it was before. The issue could not be solved in time upstream in the Apache Metron repository, so a workaround, while inferior, was inevitably the best solution.

12.3. Development process

The development process didn't go as expected. While the initial plan was to use Scrum with sprints the process changed to Kanban. The team noticed that along the way Scrum brought along too much overhead for a team of just four members. A motivation for this decision was already described in section 5.2 As was mentioned in section 5.2 the Kanban methodology worked better for the team than Scrum.

Kanban allowed the team to have a more continuous flow of development instead of trying to fit the work into week sized chunks. Using Kanban was easier than Scrum due to the issues that were faced during development. When problems arose new issues could be made and immediately worked on whereas with Scrum new issues would have to be made in the next sprint planning, halting progress on that particular problem. Furthermore Scrum gave too much overhead for a team of this size. Simply having short daily meetings with the team was found to be much easier than having weekly sprint meetings. The team found that this significantly increased the development speed as everyone is always up to date on what everyone was doing every day.

12.4. Improvements

Looking back there were a few areas in the project where improvements could have been made, although most of them were unpredictable. The issues that were encountered in the development process are consequences of the previous two discussions, namely Tensorflow and Metron. The errors that were encountered in these two systems were not derivable from the documentations. Thus these errors were completely unpredictable. But perhaps that is a good reason as well to change the development methodology. When incorporating external systems into the product, where none of the team members have any experience. It is perhaps a better idea to use a more flexible and dynamic development methodology as Kanban compared to Scrum. This will make the progression of the project more steady and notable.

13

Conclusion

Now that the end of the project has arrived, it is time to reflect on the teams accomplishment to tackle the problem stated in chapter 2.

The goals were to create a framework, which can train models given a data set and a REST-interface that serves this model, which is deployable by the Metron framework through MaaS. From these goals a list of requirements were made and prioritised through the MoSCoW method, this can be found in chapter 3. In Table 10.1, Table 10.2, Table 10.3 and Table 10.5 the requirements that have been successfully implemented have been marked.

The performance of the system was tested with a use case described in chapter 9. The system was used to create a model that could distinguish malicious certificates from benign certificates. As shown in the use case the model trained by the framework was able to distinguish the two types of certificates with an 93.5% accuracy. The system has a false positive rate of only 6%.

Overall the conclusion can be drawn that the requirements of the client and TU Delft have been met. The product is extensible to different use cases in future works.

14

Future Work

For future work there are a couple of things that still can be done. At the time of writing this report a few of the should have and could have requirements are not satisfied:

- The user should be able to decide whether the model should be exported or not
- The user could be able to specify whether error messages are returned
- The user could be able to see a progress bar of training the model
- The user could be able to specify a port number of the rest api in the config file

The team will try to satisfy these last requirements in the last week of the project after handing in the report. Besides that the team will try to further increase the test score on both components of the system.

There are other features that could be added in the long term. The first is adding the possibility to pre-process the data in the neural network framework, as mentioned in subsection 8.1.3. This will allow a user to process the input data with column based or single item based calculations via custom functions. This can speed up the model development phase for the user and makes the framework more interesting to use. Pre made functions could also be added, like transformation functions, entropy functions or basic clustering functions.

The second long term feature that should be added is the addition of other machine learning models. The first model that would be added is a neural network regression model, as it is similar to the neural network classification model that is currently used. Other models could also be added, like support vector machines, random forest trees or clustering models.

As a third feature instead of a command line interface, a graphical user interface could be built around the neural network framework.

Another feature that could be added is the use of graphical benchmark information beside numerical benchmarks used right now. This could be added to the GUI previously mentioned.

A

Research report

A.1. Introduction

This research report describes the research that was done in the first weeks of the project. The research phase can be divided into two parts: phase 1 and phase 2. This was not intentional, but after the first two weeks, it became clear that the proposed solution would not fit the Bachelor End Project requirements of the TU Delft. The project was changed and a new research phase started. The two phases combined took up almost four weeks of the project, therefore, they are both included in this report.

Note that this research report should be read in conjunction with the final report. Some parts which were duplicate in the research report and in the final report have been removed from the research report.

A.2. Research phase 1

A.2.1. Client requirements

The research phase started off with interviews with the client. The client specified several requirements regarding the final product:

- The data that comes from the optical tap is traffic that enters or leaves Intermax's network, so no internal network data is available
- The data is not allowed to leave Intermax's network
- The system should be scalable
- The system should be delivered as a REST API that will be incorporated into Apache Metron[1] through the Model as a Service (MaaS)[2] functionality

The client wanted the system to detect whether any of their system were compromised and whether possible data exfiltration was taking place. In addition, they wanted the system to detect incoming threats, but this was not a hard requirement. The focus would thus be to analyze the network data and highlight anomalous traffic to detect data exfiltration and threats.

A.2.2. Threat detection methods

The next step was to research which threat detection methods already exist for detecting data exfiltration. This was done through reading academic papers and discussion with the lead security engineer of Intermax, and the TU Delft coach, who does research about cyber threat intelligence. It became clear that detecting data exfiltration and threats on networks is a current challenge for the cyber intelligence industry [23] and a great amount of research

[1] http://metron.apache.org/
[2] https://metron.apache.org/current-book/metron-analytics/metron-maas-service/index.html

exists on this topic. Three surveys on anomaly detection [2], [7], [15] and three additional papers [26], [14], [16] were consulted to create an overview of threat detection methods. This overview can be found in Appendix B. From these, two general approaches were proposed: signature matching and anomaly detection using baselines.

Signature matching

Signature matching relies on blacklists which contain malicious certificates, IP addresses and other network data. The idea is to download several blacklists from the internet and check the incoming network data against these blacklists. If the data is present in the blacklist, it will be marked as malicious by giving the data packet a high threat score. The blacklists will be updated regularly to ensure that the newest threats will be detected as well.

It is also possible to do signature matching based on whitelists which contain trusted certificates, IP address and other network data. If a data packet is matched with an entry in the whitelist, it will be approved and a very low threat score will be given. Because there is no labelled data available, it will be quite hard to decide whether something should be in the whitelist or not. Therefore, if a whitelist implementation is chosen, the client will have to create the whitelist manually.

Anomaly detection using baselines

Anomaly based detection has several steps:

1. Analyse data that does not contain potential threats
2. Derive from this data what is normal in order to create a baseline
3. Compare new data with the baseline and give threat score based on the deviation from the baseline
4. Continuously update the baseline using steps 1 and 2

These steps are general for an approach which uses a baseline. The baseline can however be derived from different types of data. The proposal was made to create baselines for the following types of data:

- Packet size
- Packet frequency
- Port frequency
- Protocol patterns
- Packet header fields
- IP addresses

For the packet size, packet frequency and port frequency approach, distribution models would be generated based on the historical data of the network. The challenge here is deciding what the windows for the model and for the sample will be as there is a high volume of incoming data. Using the generated models, a threat score would be derived by comparing the variances and the means of the model and sample.

The protocol patterns approach is similar, but here the frequency of the different protocol calls is used to create a baseline for what is 'normal'. Consequently the sample will be compared to the baseline and a threat score will be given.

A bit more far-fetched approach was to create a wavelet based on the IP-addresses to simulate an visual representation of the network traffic [14]. Based on these, anomalies could be derived. However, this method was complex and less accurate, so this was a consideration for later if enough time was left.

A.2.3. Apache Metron and reconsideration

After having an overview of possible solutions for detecting data exfiltration and threats, the next step was to research how to implement this in the Apache Metron framework. Apache Metron is a security analytics framework created by the Apache Software Foundation and Intermax is developing a Security Operations Center based on this framework. The details of Apache Metron can be found in chapter 6.

The initial idea was that Apache Metron only regulates the data pipeline, but it actually has more features. This includes creating profiles to store data which can be used for analysis, meaning that the proposed solutions could easily be implemented by configuring Apache Metron and REST API might not even be needed anymore. Even though this might seemed as a good solution, it meant that the project would be reduced from creating code to simply modifying configuration files of Metron. This would not meet the level of software engineering that is required from the TU Delft. Something had to be changed to increase the software engineering aspect of the project, marking the start of research phase 2.

A.3. Research phase 2

A.3.1. TU Delft software requirements

As the level of software engineering that is required in the project is not specified clearly, the team decided to discuss the problem with one of the Bachelor End Project coordinators, Otto Visser, and the client adviser. Otto indicated that creating some sort of framework to create models for the REST API would increase the software engineering aspect of the project. In addition, the weights of the rubric for the Bachelor End Project could be slightly adjusted to give more weight to the research part of the project.

A.3.2. Adapted client requirements

Event though Intermax initially just wanted a REST API and corresponding models, they accepted the idea of creating a framework. The Lead Security engineer proposed that a proof of concept for detecting command and control using SSL/TLS data would be created using the framework. The solution would not yet have to work with the 500 Mbit/s data stream, but it should be scalable for the future. The other requirements would still hold.

A.3.3. Possible solutions

The team started researching again and now focused on the command and control proof of concept. This was again done by going through academic papers. The paper from Gardiner et al. [8] provided a good introduction into command and control schemes. The paper gave some good insights into the possibilities, but additional papers were used to get to a fitting solution for the problem. Malware is often used in command and control schemes and below, different approaches for detecting malware are outlined. The research eventually points to using SSL/TLS as a good solution.

Data analysis approach

In the paper by [4], several malware detection methods using data analysis are mentioned. The method proposed by [3] focuses on finding structural differences between benign and malicious certificates and the method proposed by [5] use IPv4 addresses. Dong does however already note that these kind of detection methods require centralized network services[4]. This would limit the data available for creating models to the network data of Intermax. An approach using additional data would increase detection rates as publicly known information about malware can be taken into account.

Machine learning approaches

As noted by both [4] and [25], machine learning approaches are becoming more and more popular in detecting malware as they produce higher success rates. [20] reported an accuracy rate of 92% and [6] used machine learning to reach 98% accuracy.

Machine learning algorithms thus show to be quite promising in detecting malware. Creating a framework in which machine learning models can be created and trained would provide a way to generate different kinds of models which can be updated and adjusted according to the user's needs.

Using SSL/TLS data

For detecting suspicious network activity, several methods exist. Signature matching methods which have been discussed in section A.2 are ruled out because of the TU Delft requirements for the project. [25] presents multiple machine learning algorithms such as streaming

analytics [18], gradient boosting [19] and online incremental learning [17]. These methods do however require to analyze websites upfront, increasing the evaluation process greatly.

Recurrent neural networks (RNN) are another form of machine learning. RNNs recognize patterns based on sequential dependencies as time series variables or text data [25]. Apache Metron will however look at individual data points and spawns independent instances of the API to analyze the data points. Data can thus not be combined to look at larger time windows or time series, therefore RNNs are also not an option.

The paper by [25] presents a deep neural network which uses the features of SSL certificates to detect malware and command and control schemes. These features (Table G.1) are extracted from the SSL certificates and are used in the deep neural network to classify the certificates. This method achieves models with an accuracy rate of roughly 95%. In addition, the method can be applied to single SSL certificates, which are single data points, and it is thus very applicable to the use case at hand. With speed in mind, the decision has been made to first create a DNN using only the Boolean and Splunk category features, as these can be processed faster than the other two categories. The DNN will however need to be trained with a labelled data set. The data set will either be acquired from existing data sets on the internet or will be created manually through a script which scrapes benign and malicious SSL certificates from the internet. The choice of programming languages and frameworks are discussed in chapter 7.

B

Overview of possible solutions

B.1. Anomaly detection

- Statistical methods
  - PCA
  - Wavelet
  - Covariance matrix
- Rule based
- Distance based (Information theory)
  - Entropy
- Profiling method
  - PHAD (packet header anomaly detection)
- Classification
  - Naïve bayesian
  - Replicator neural networks
  - Artificial neural networks
  - Unsupervised support vector machines
- Clustering
  - K-means
  - k-NN
  - Subspace clustering
- Evolutionary
  - Artificial immune system
  - Genetic algorithms
  - Particle swarm optimization SVM

B.2. Misuse detection

- Signature matching
  - TLS certificate matching
  - IP address matching
- Rule based
- State-transition analysis
- Data mining based
  - Association rules
  - Standard data mining

C

Project Planning

Below, you will find the project planning that was created at the start of the project. This was done roughly after the first talks with the client and when there was a basic understanding of the problem and possible solutions. The dates indicate the start of the week.

Week 1 (22-04-2019)

- Initial interviews with the client
- Do research on how to detect threats on networks
- Gather several threat detection methods which could serve as part of the solution

Week 2 (29-04-2019)

- Dive deeper into the different threat detection methods
- Decide on the feasibility of each method in relation to the use case
- Select methods which will be incorporated in the final software system
- Start with additional research on implementation

Week 3 (06-05-2019)

- Work out how the found methods can be applied to the use case
- Create architecture design and decide upon which languages/frameworks to use
- Begin with setting up the framework for the software system

Week 4 (13-05-2019)

- Finish framework for the solution
- Start implementing threat detection methods

Week 5-8 (20-05-2019)

- Complete the software system including testing

Week 9 (17-06-2019)

- Applying final tweaks to the software system
- Focus on finishing the report draft

Week 10 (24-06-2019)

- Finish report
- Start on presentation including demo

Week 11 (01-07-2019)

- Finish presentation
- Conclude the project

D

Manuals

Manual.md

BEP Neural Network Framework Manual

Summary

The Neural Network Framework is a program that enables users to easily construct a neural network by editing a config file. The input data location/format is specified by the user in this config file and upon running the program it trains the neural network and gives a score as a result. The model is then saved for later use by the REST API server.

Build instructions

Run the following commands:

- pip install -r requirements.txt
- python setup.py bdist_wheel

For tests:

- pytest --cov=src --cov-branch -vv

Run instructions

Requirements

- Python 3.6+
- CPU with AVX instructions (required by Tensorflow, see here)

Configuration

To run this neural network framework you need to first configure the config file for the Neural Network Framework. An example is provided in `Resources/cert_config_2.json`. The following fields are required in the JSON config:

- `model_name` The name of the model
- `train_csv_path` The file path of the training data in csv-format
- `test_csv_path` The file path of the test data in csv-format
- `model_base_dir` The directory to store the model in
- `input` An object with the inputs for the neural network. Each element in this object has a name as key, the value is another object with two fields:
  - `index` The index of the column in the csv file (index starts at 0)
  - `type` The value type of the column (must be a number so either `float` or `int`)
- `output` An object with the output for the neural network. The output of the neural network. Must contain a single field so, the value of this field contains two values:
  - `index` The index of the output feature in the csv file (index starts at 0)
  - `type` The type of the output layer, must be `int` for classification and `float` for regression.
- `model` Type of neural network, either `classification` or `regression`. Currently only `classification` is implemented
- `layers` An array of integers. The order is from input layer to output layer, so an array of `[10, 5, 3]` is a neural network that looks like: `Input layer -> 10 node layer -> 5 node layer -> 3 node layer -> output layer`.

Running the prgoram

To run the program you need to run the `src/nnf.py` with the first program argument being the path to the config file.

Manual.md

BEP Metron REST API Manual

Summary

The REST API only has one endpoint: `/apply`. This endpoint is called with query parameters that form the input data. These query parameters are transformed into valid inputs for the neural network. After the transformation they are applied in the neural network after which a result is returned from the network which goes back to Metron.

Build instructions

Run the following commands:

- pip install -r requirements.txt
- python setup.py bdist_wheel

For tests:

- pytest --cov=src --cov-branch -vv

Run instructions

To simply run the server for local use run the `src/main.py` with the program argument being the path to the REST API config file.

Deploy instructions

Requirements

- Python 3.6+
- Apache Metron 0.7.1+
- A model trained by the Neural Network Framework project on the local filesystem

REST API config

The REST API needs a config file to know where to find the models and how to transform the input queries to valid input for the neural network. An example of this config file is: `Resources/cert_config_2.json`.

The config requires the following fields in JSON format:

- `model_name` The name of the model, this must correspond to the name of the model as given in the Neural Network Framework project config
- `model_base_dir` The directory where the model directory is stored, so if the model is called `model_example` and stored in `/home/user/models/model_example` the value would be `/home/user/models`.
- `input` An object that describe how to get the input for the neural network from the query parameters. Each element in the object corresponds to the input of the neural network. The name of the element **has** to correspond to the name of the feature as given in the Neural Network Framework project. The value of the element is a `function` object which has two elements:
  - `name` The name of the custom python function to call in `src/custom_code.py`
  - `params` An array of pairs, these are the parameters that are passed to the function identified by `name`. These parameters are passed to the function in the **same order as they are defined**. A pair can look like `["key", "some_param"]` or `["var", "some_value]`. A pair with `key` means that the second value is the name of the query parameter that should be passed to the function, so for a query parameter `qparam1=20`, `["key", "qparam1"]` would result in `20` being passed to the function. A pair with `var` is a static variable, this can either be a number, a boolean or a string, so `["var", 30]` would result in `30` being passed to the function.

- An example of an `input` element is given below

```
"feature_1": {
    "function": {
        "name": "to_feature_1",
        "params": [["key", "query_param_name_1"],["key", "query_param_name_2"],["var", 2],["var", "some_string"]]
    }
},
```

Custom code

As mentioned in the previous section there is a `custom_code.py` file where function can be defined to transform the query parameters to valid neural network inputs (numbers).

An example function for converting a something to an int:

```
def str_to_int(first_arg):
    return int(first_arg)
```

As you can see only the function needs to be defined, there is no need for imports etc. It is immediately usable by the REST API. The only thing to bear in mind is to not use common method names such as `function` or `main` as these will not work. Two functions of the same name with a different amount of arguments will also not work.

Folder restructuring

Apache Metron flattens all files that are copied to HDFS, which means that all fils are put in the root directory of the container. To solve this issue we relocate the model files from Tensorflow that **have** to be in a subfolder. In the `src/restructure_folder.py` script the `model_name` needs to have the value of the model name as specified in the REST API config and `maas_deploy.sh` script.

Deploy script

To deploy this MaaS service run the script: `Resources/maas_deploy.sh`. This script has the following parameters that **have** to be modified inside the script itself:

- Metron variables:
  - `METRON_HOME_PATH` The path to the Apache Metron installation
- MaaS variables:
  - `MODEL_NAME` The name of the model
  - `MEM_IN_MB` The memmory in MB to assign to the container
  - `VERSION` The version of the model
  - `NUM_INSTANCES` The amount of instances to run
  - `ZOOKEEPER_HOST` The Zookeeper hostname and port
- Model paths:
  - `LOCAL_SRC_PATH` The directory of the source code to deploy (this must be the `src` folder path of this repo)
  - `LOCAL_MODEL_PATH` The directory where the models reside on the local filesystem, these will be copied to HDFS
  - `LOCAL_CONFIG_PATH` The path of the config file, this will be copied to HDFS

Apache Metron config

In Apache Metron the config needs to be modified such that it calls the REST API. In the Stellar language this function needs to be called in order to make a request to the REST API:

```
"MAP_GET('score', MAAS_MODEL_APPLY(MAAS_GET_ENDPOINT('maas_name'), {'qparam1' : value1, 'qparam2': value2}))"
```

In this request `maas_name` is the name of the model as specified in the `maas_deploy.sh` script.

E

Configuration files

E.1. SSL certificate model

E.1.1. Neural Network Framework

```
{
  "model_name": "c2_alexa_cert_model",
  "train_csv_path": "<DATADIR>\\c2_alexa_data.csv",
  "test_csv_path": "<DATADIR>\\c2_alexa_data_test.csv",
  "model_base_dir": "<EXPORTDIR>",
  "input": {
    "ShannonCN_nom": 10,
    "IssuerLength_nom": 11,
    "IssuerElements_nom": 12,
    "SubjectLength_nom": 13,
    "SubjectElements_nom": 14,
    "ExtensionNumber_nom": 15,
    "ExtensionLength_nom": 16,
    "Time_Interval_nom": 17,
    "Subject_Issuer_nom": 9
  },
  "output": {
    "Malicious": 18
  },
  "model": "classification",
  "layers": [
    20,15, 10, 5
  ]
}
```

E.1.2. REST API

config.json:

```
{
    "model_name": "c2_alexa_cert_model",
    "model_base_dir" : "<EXPORTDIR>",
    "input": {
      "IssuerLength_nom": {
        "function": {
          "name": "get_issuer_len",
          "params": [["key", "issuer"], ["var", 3], ["var", 180]]
        }
      }, "IssuerElements_nom": {
        "function": {
          "name": "get_issuer_element_len",
          "params": [["key", "issuer"], ["var", 1], ["var", 8]]
        }
      }, "SubjectLength_nom": {
        "function": {
          "name": "get_subject_len",
          "params": [["key", "subject"], ["var", 3], ["var", 294]]
        }
      }, "SubjectElements_nom": {
        "function": {
          "name": "get_subject_elements_len",
          "params": [["key", "subject"], ["var", 1], ["var", 14]]
        }
      }, "ShannonCN_nom": {
        "function": {
          "name": "to_shannon_cn",
          "params": [["key", "subject"], ["var", 0], ["var", 4.69]]
        }
      }, "ExtensionLength_nom": {
        "function": {
          "name": "get_extension_len",
          "params": [["key", "extension_size"], ["var", 0], ["var", 193]]
        }
      }, "ExtensionNumber_nom": {
        "function": {
            "name": "get_extensions_num",
            "params": [["key", "extension_count"], ["var", 0], ["var", 12]]
          }
      }, "Time_Interval_nom": {
        "function": {
          "name": "get_time_interval",
          "params": [["key", "not_valid_before"], ["key", "not_valid_after"],
            ↪  ["var", 3.40], ["var", 10.65]]
        }
      }, "Subject_Issuer_Equal": {
        "function": {
          "name": "sub_iss_equal",
          "params": [["key", "issuer"], ["key", "issuer"]]
        }
      }
    }
}
```

custom_code.py:

```
import datetime
import math
import re

def normalize(a, min_val, max_val):
    """
    Normalization function
    :param a: Value to normalize
    :param min_val: Min value
    :param max_val: Max value
    :return: The normalized value
    """
    return (a - min_val) / (max_val - min_val)


def entropy(string):
    """
    Calculates the Shannon entropy of a string
    :param string: The input string
    :return: Shannon entropy of the string
    """
    # get probability of chars in string
    prob = [float(string.count(c)) / len(string) for c in dict.fromkeys(list(string))]

    # calculate the entropy
    string_entropy = - sum([p * math.log(p) / math.log(2.0) for p in prob])

    return string_entropy


def get_sub_iss_element_len(sub_or_iss):
    """
    Get the sub or issuer element count
    :param sub_or_iss: Subject or issuer
    :return: The amount of elements in subject or issuer
    """
    if sub_or_iss == "":
        return 0
    split = sub_or_iss.split(",")
    return len(split)


def sub_iss_value_len(sub_or_iss):
    """
    Get the total length of the subject or issuer values
    :param sub_or_iss: Subject or Issuer
    :return: The length of all values in the subject or issuer
    """
    if sub_or_iss == "":
        return 0
    split_comma = sub_or_iss.split(",")
    count = 0
    for s in split_comma:
        split_equals = s.split("=")
        if len(split_equals) == 2:
            count += len([1])
    return count


```

```
60  def to_shannon_cn(subject, min_val, max_val):
61      cn = re.search("CN=(.+?),", subject)
62      if cn is not None:
63          return normalize(entropy(cn.group(1)), min_val, max_val)
64      # Return an entropy of zero when there is no CN
65      return 0
66
67
68  def get_issuer_len(issuer, min_val, max_val):
69      return normalize(sub_iss_value_len(issuer), min_val, max_val)
70
71
72  def get_issuer_element_len(issuer, min_val, max_val):
73      count = get_sub_iss_element_len(issuer)
74      return normalize(count, min_val, max_val)
75
76
77  def get_subject_len(subject, min_val, max_val):
78      return normalize(sub_iss_value_len(subject), min_val, max_val)
79
80
81  def get_subject_elements_len(subject, min_val, max_val):
82      count = get_sub_iss_element_len(subject)
83      return normalize(count, min_val, max_val)
84
85
86  def get_extensions_num(extension_count, min_val, max_val):
87      return normalize(int(extension_count), min_val, max_val)
88
89
90  def get_extension_len(extension_size, min_val, max_val):
91      return normalize(int(extension_size), min_val, max_val)
92
93
94  def get_time_interval(not_valid_before, not_valid_after, min_val, max_val):
95      day_before = datetime.datetime.fromtimestamp(int(not_valid_before))
96      day_after = datetime.datetime.fromtimestamp(int(not_valid_after))
97      return normalize(math.log((day_after - day_before).days), min_val, max_val)
98
99
100 def sub_iss_equal(subject, issuer):
101     return int(subject == issuer)
```

Original project description

Below, you will find the original project description which was posted on BEPSys (https://bepsys.ewi.tudelft.nl/course_editions/7/projects/333):

Real-Time Threat Detection Through Network Analysis

Achtergrond

Intermax ontwerpt, implementeert en beheert als 'cloud-provider' kritieke IT-infrastructuren van klanten in onder meer de zorg, publieke sector en Fintech. Als cloud-provider beschikt Intermax over een groot aantal systemen. Hieronder vallen zowel virtuele (VMWare) als fysieke hosts dan wel netwerk-apparatuur. Als aanvulling op het leveren van veilige en betrouwbare infrastructuren wordt er een Security Operations Center ontwikkeld. Dit SOC moet de data die voortkomt uit deze systemen analyseren en hiermee security incidenten detecteren om Data-driven decision making mogelijk te maken. Het doel van dit project is het aanvullen van dit SOC middels het analyseren en classificeren van netwerkinformatie.

Projectbeschrijving

Dit project focust zich op het analyseren van netwerk-gerelateerde data. Intermax heeft een tap gezet op één van haar core-routers. Met deze tap worden alle verbindingen die Intermax en haar klanten met de buitenwereld maken (500 Mbit/s) opgeslagen in .PCAP formaat en verwerkt in een centrale Elasticsearch database. Het ontwikkelde SOC (gebaseerd op Apache Metron) verwacht een REST-interface welke nieuwe observaties opneemt en een classificatie hiervan teruggeeft. Het is de bedoeling dat in dit project modellen worden ontwikkeld om de beschikbare netwerk-informatie te analyseren en een mate van risico te bepalen. Omdat nieuwe datapunten in hoog tempo binnenkomen is het ontwikkelen van een real-time oplossing een uitdaging voor Intermax. Behalve een REST-interface voor het classificeren van nieuwe observaties zijn er nog geen ontwerpkeuzes gemaakt en hierin is volledige vrijheid. Het project wordt als succesvol beschouwd indien er een werkend prototype met REST-interface beschikbaar is waar observaties geclassificeerd worden. Hierbij is het belangrijk dat de ontworpen software schaalbaar is en de modellen 'bijleren' van nieuw verkregen data.

Gerelateerde Links:

Intermax: https://intermax.nlMetron:http://metron.apache.org/
Apache spark ('Big-data engine' beschikbaar in Metron): https://spark.apache.org/
Threat Intel: https://cymon.io/

Other information

In verband met het in aanraking komen met zeer vertrouwelijke data moet er een NDA ondertekend worden.

About Intermax

Established in 1994, The Intermax Group consists of Intermax Cloudsourcing, Bizway, Guardian360, NFIR and Intermax Datascience. As a group these companies keep complex cloud infrastructures and advanced security services safe and available 24/7. They work for organizations that cannot do without a safe and reliable IT environment. The Intermax Group also is your main point of contact for forensic IT services and employs a total of 110 people; all specialists in the fields of cloud technology and cyber security. Intermax also is the inventor of the Dutch National Anti DDOS Scrubbing Center and was the first party in Europe to integrate a cyber insurance into its provision of services. The Intermax Group is 100% Dutch and fully independently financed.

G

List of SSL features

Feature Name	Description	Category
SubjectCommonNameIp	Indicates if CN is an IP address instead of domain	Boolean
Is_extended_validated	Indicates if certificate is extended validated	Boolean
Is_organization_validated	Indicates if certificate is organization validated	Boolean
Is_domain_validated	Indicates certificate is domain validated	Boolean
SubjectHasOrganization	Indicates if subject principal has O field	Boolean
IssuerHasOrganization	Indicates if issuer principal has O field	Boolean
SubjectHasCompany	Indicates if subject principal has CO field	Boolean
IssuerHasCompany	Indicates if issuer principal has CO field	Boolean
SubjectHasState	Indicates if subject principal has ST field	Boolean
IssuerHasState	Indicates if issuer principal has ST field	Boolean
SubjectHasLocation	Indicates if subject principal has L field	Boolean
IssuerHasLocation	Indicates if issuer principal has L field	Boolean
Subject_onlyCN	Indicates if subject principal has only CN field	Boolean
Subject_is_com	Indicates if subject CN is a ”.com” domain	Boolean
Issuer_is_com	Indicates if issuer CN is a ”.com” domain	Boolean
HasSubjectCommonName	Indicates if CN is present in subject principal	Boolean
HasIssuerCommonName	Indicates if CN is present in issuer principal	Boolean
Subject_eq_Issuer	Boolean indicating if Subject Principal = Issuer Principal	Boolean
SubjectElements	Number of details present in subject principal	Splunk
IssuerElements	Number of details present in issuer principal	Splunk
SubjectLength	Number of characters of whole subject principal string	Splunk
IssuerLength	Number of characters of whole issuer principal string	Splunk
ExtensionNumber	Number of extensions contained in the certificate	Splunk
Selfsigned	Indicates if certificate is self signed	SOC
Is_free	Indicates if the certificate is free generated	SOC
DaysValidity	Calculated days between not before and not after days	SOC
Ranking_C	Calculated ranking of domain based on domain ranking	SOC
SubjectCommonName	Calculated character entropy in the subject CN	Text
Euclidian_Subject_Subjects	Calculated euclidean distance of subject among all subjects	Text
Euclidian_Subject_English	Calculated euclidean distance of subject characters among English characters	Text
Euclidian_Issuer_Issuers	Calculated euclidean distance of issuer among all issuers	Text
Euclidian_Issuer_English	Calculated euclidean distance of issuer characters among English characters	Text
Ks_stats_Subject_Subjects	Kolmogorov-Smirnov statistics for subject in subjects	Text
Ks_stats_Subject_English	Kolmogorov-Smirnov statistic for subject in English characters	Text
Ks_stats_Issuer_Issuers	Kolmogorov-Smirnov statistics for issuers in issuers	Text
Ks_stats_Issuer_English	Kolmogorov-Smirnov statistic for issuer in English characters	Text
Kl_dist_Subject_Subjects	Kullback-Leiber Divergence for subject in subjects	Text
Kl_dist_Subject_English	Kullback-Leiber Divergence for subject in English characters	Text
Kl_dist_Issuer_Issuers	Kullback-Leiber Divergence for issuer in Issuers	Text
Kl_dist_Issuer_English	Kullback-Leiber Divergence for issuer in English characters	Text

Table G.1: Features from the Torroledo et al. study[25]

Table G.2: Table of 43 feature used in the first models

Feature name	Description	Category
IssuerO	Indicates if issuer O field is set	Boolean
IssuerCO	Indicates if issuer CO field is set	Boolean
IssuerST	Indicates if issuer ST field is set	Boolean
IssuerL	Indicates if issuer L field is set	Boolean
IssuerCN	Indicates if issuer CN field is set	Boolean
IssuerDotcom	Indicates if issuer CN is a '.com' domain	Boolean
IssuerLength	Number of characters of whole issuer principal string	Integer
IssuerElements	Number of details present in issuer principal	Integer
SubjectO	Indicates if subject O field is set	Boolean
SubjectCO	Indicates if subject CO field is set	Boolean
SubjectST	Indicates if subject ST field is set	Boolean
SubjectL	Indicates if subject L field is set	Boolean
SubjectCN	Indicates if subject CN field is set	Boolean
SubjectDotcom	Indicates if subject CN is a '.com' domain	Boolean
SubjectLength	Number of characters of whole subject principal string	Integer
SubjectElements	Number of details present in subject principal	Integer
SubjectIP	Indicates if CN is an IP address instead of domain	Boolean
ExtensionNumber	Number of extensions contained in the certificate	Integer
Extensions	One hot encoding for each of the 25 extension's presence or absence in the certificate	Boolean

Table G.3: Table of 7 and 9 features

Feature name	Description	Category
IssuerLength	Number of characters of whole issuer principal string	Integer
IssuerElements	Number of details present in issuer principal	Integer
SubjectLength	Number of characters of whole subject principal string	Integer
SubjectElements	Number of details present in subject principal	Integer
ExtensionNumber	Number of extensions contained in the certificate	Integer
ExtensionLength	Number of characters of all extension strings	Integer
ShannonCN	Shannon entropy of subject CN	Float
Time_Interval	natural log of Time interval between not before and not after in days	Float
Subject_Issuer_Equal	Indicates if the subject and issuer principals are equal	Boolean

H Weekly logs

H.1. Process overview

In this chapter, you will find a description of the process in each week and how it relates to the initial planning. The dates indicate the start of the week.

H.1.1. Week 1 (22-04-2019)

In the first week, we had the initial interviews with the client and started researching the problem. We found out that the problem and possible solutions were rather complex and could be hard to implement. This was because we had mainly looked at academic examples of threat detection but these examples are usually not feasible for businesses.

Overall, we are still on track.

H.1.2. Week 2 (29-04-2019)

We started off the week with a meeting with the lead security engineer of Intermax. We discussed several threat detection methods, but after some additional research we came to the conclusion that these would not fit within the scope of the project.

The rest of the week, we spent time figuring out how Apache Metron works and reading the papers that were given to us by our supervisor. At the end of the week, we had a meeting with multiple members of Intermax to discuss our process and approach to the problem. The people of Intermax were enthusiastic about our idea, but they preferred us to start working on a simple blacklisting method so that we were sure to deliver something.

Compared to our initial planning, we are still on track.

H.1.3. Week 3 (06-05-2019)

At the start of the week, we had a meeting with the client. We found out that Apache Metron actually already has built-in functionality for analysing data streams and checking them against blacklists. In addition, the client had decided that they were going to implement the blacklist method themselves using Apache Metron.

In order to ensure that there is a sufficient amount of software engineering in our project, we decided to do something with machine learning. We talked with Otto Visser about our idea and he suggested that we should create something like a framework so that the created product would live up to the software engineering requirements.

We lost quite a lot of time in this week because we had to reorganise the project together with the client and the TU Delft. In addition, we only got access to the Intermax GitLab on Friday, which limited our capabilities to work on the project before that.

At this point, we are starting to get a bit behind the initial schedule.

H.1.4. Week 4 (13-05-2019)

In week 4, we made quite some progress with the setup of the systems. Jim and Kabilan did a lot of the basic set up for both of the API and the Neural Network, this was delayed because

we had to wait for Intermax to assign a runner to our project. Just and Jim managed to get TensorFlow up and running.

Pravesh was absent for almost the whole week due to illness. We are now roughly a week behind schedule, as the setup should have been finished in the previous week.

H.1.5. Week 5 (20-05-2019)

This week we further optimized the CI. We did however run into a problem with testing TensorFlow on the CI. TensorFlow uses AVX instructions to increase its performance, but the runner of the CI was not compatible with these instructions. Therefore we had to disable the test stage for TensorFlow and run the tests locally.

We also wrote two scripts to collect certificates from the internet which can be used for training our neural network. We have gathered over 17,000 malicious, phishing and benign certificates which can be used for training purposes.

Compared to the initial planning, we are a little more than a week behind. The framework is almost finished and we haven't started with the actual threat detection yet.

H.1.6. Week 6 (27-05-2019)

The neural network framework is pretty much finished. We started with training some models and applying them. To our surprise, we had very high accuracies (> 95%). We also focussed on making our code ready for the SIG upload. The certificate scraper files are not interesting for the SIG and test coverage has to increase. We also managed to implement field parsing in Metron, we still struggle with parsing multiple fields with Bro.

We are now a bit more on track. A large part of the core of the system has been finished. If the Bro issues are resolved, we are ready to start testing the detection of different types of threats.

H.1.7. Week 7 (03-06-2019)

This week was all about Metron. Jim and Just finally got the Bro and Metron parsing working after a lot of trial, error and restarting of services. After the parsing part was over they could start with the first certificate predictions. First the custom functions had to be written to parse the data input. Soon it became clear that ones again Metron was not working as expected and it took a couple of days to debug the system line for line and in the end get it working. At the same time Pravesh and Kabilan started with the final report. At the end of week 7 we finally got the first real certificate prediction results from metron, and they were pretty good!

H.1.8. Week 8 (10-06-2019)

This week the final report was the most important. While Pravesh and Kabilan were still writing, Just was gathering results and creating graphs from the different models. Jim was gathering results from the metron framework and performing a couple of tweaks. At the end of week 8 we send the first draft version to our TU Delft coach.

H.1.9. Week 9 (17-06-2019)

On Monday morning we had only 7 days left before the final report had to be handed in. This week all for were full time working on the report. On Wednesday we had a meeting with Rene to discuss the last things that could be changed in week 10. On Thursday we had a meeting with our TU coach to discuss the draft version of the report.

Software Improvement Group Code Quality Evaluation

I.1. Feedback REST API

De code van het systeem scoort 4.1 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code bovengemiddeld onderhoudbaar is. De hoogste score is niet behaald door lagere scores voor Unit Interfacing en Unit Complexity.

Voor Unit Interfacing wordt er gekeken naar het percentage code in units met een bovengemiddeld aantal parameters. Doorgaans duidt een bovengemiddeld aantal parameters op een gebrek aan abstractie. Daarnaast leidt een groot aantal parameters nogal eens tot verwarring in het aanroepen van de methode en in de meeste gevallen ook tot langere en complexere methoden. Dit kan worden opgelost door parameter-objecten te introduceren, waarbij een aantal logischerwijs bij elkaar horende parameters in een nieuw object wordt ondergebracht. Dit geldt ook voor constructors met een groot aantal parameters, dit kan een reden zijn om de datastructuur op te splitsen in een aantal datastructuren. Als een constructor bijvoorbeeld acht parameters heeft die logischerwijs in twee groepen van vier parameters bestaan, is het logisch om twee nieuwe objecten te introduceren.

Voorbeelden in jullie project:
- `Config.__init__(model_name,input_dict,output,model,model_dir)`

Voor Unit Complexity wordt er gekeken naar het percentage code dat bovengemiddeld complex is. Dit betekent overigens niet noodzakelijkerwijs dat de functionaliteit zelf complex is: vaak ontstaat dit soort complexiteit per ongeluk omdat de methode te veel verantwoordelijkheden bevat, of doordat de implementatie van de logica onnodig complex is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, makkelijker te testen is, en daardoor eenvoudiger te onderhouden wordt. Door elk van de functionaliteiten onder te brengen in een aparte methode met een beschrijvende naam kan elk van de onderdelen apart getest worden, en wordt de overall flow van de methode makkelijker te begrijpen. Bij grote en complexe methodes kan dit gedaan worden door het probleem dat in de methode wordtd opgelost in deelproblemen te splitsen, en elk deelprobleem in een eigen methode onder te brengen. De oorspronkelijke methode kan vervolgens deze nieuwe methodes aanroepen, en de uitkomsten combineren tot het uiteindelijke resultaat.

Voorbeelden in jullie project:
- `config_validator.py:__check_input(json)`
- `config_validator.py:__check_output(json,model)`

De aanwezigheid van testcode is in ieder geval veelbelovend. De hoeveelheid testcode ziet er ook goed uit, hopelijk lukt het om naast toevoegen van nieuwe productiecode ook nieuwe tests te blijven schrijven.

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

I.2. Feedback Neural Network Framework

De code van het systeem scoort 3.8 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code marktgemiddeld onderhoudbaar is. We zien Unit Interfacing en Unit Size vanwege de lagere deelscores als mogelijke verbeterpunten.

Voor Unit Interfacing wordt er gekeken naar het percentage code in units met een bovengemiddeld aantal parameters. Doorgaans duidt een bovengemiddeld aantal parameters op een gebrek aan abstractie. Daarnaast leidt een groot aantal parameters nogal eens tot verwarring in het aanroepen van de methode en in de meeste gevallen ook tot langere en complexere methoden. Dit kan worden opgelost door parameter-objecten te introduceren, waarbij een aantal logischerwijs bij elkaar horende parameters in een nieuw object wordt ondergebracht. Dit geldt ook voor constructors met een groot aantal parameters, dit kan een reden zijn om de datastructuur op te splitsen in een aantal datastructuren. Als een constructor bijvoorbeeld acht parameters heeft die logischerwijs in twee groepen van vier parameters bestaan, is het logisch om twee nieuwe objecten te introduceren.

Voorbeelden in jullie project:
- `Config.__init__(model_name,train_csv_path,test_csv_path,input,output,model,layers,model_export_dir)`

Bij Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Dit kan verschillende redenen hebben, maar de meest voorkomende is dat een methode te veel functionaliteit bevat. Vaak was de methode oorspronkelijk kleiner, maar is deze in de loop van tijd steeds verder uitgebreid. De aanwezigheid van commentaar die stukken code van elkaar scheiden is meestal een indicator dat de methode meerdere verantwoordelijkheden bevat. Het opsplitsen van dit soort methodes zorgt er voor dat elke methode een duidelijke en specifieke functionele scope heeft. Daarnaast wordt de functionaliteit op deze manier vanzelf gedocumenteerd via methodenamen.

Voorbeelden in jullie project:
- `nnf.py:custom_prediction(data_file,config)`
- `nnf.py:import_config_file(config_file_location)`
- `config_validator.py:__check_assigned_values(json)`

De aanwezigheid van testcode is in ieder geval veelbelovend. De hoeveelheid tests blijft nog wel wat achter bij de hoeveelheid productiecode, hopelijk lukt het nog om dat tijdens het vervolg van het project te laten stijgen. Op lange termijn maakt de aanwezigheid van unit tests je code flexibeler, omdat aanpassingen kunnen worden doorgevoerd zonder de stabiliteit in gevaar te brengen.

Over het algemeen is er dus nog wat verbetering mogelijk, hopelijk lukt het om dit tijdens de rest van de ontwikkelfase te realiseren.

J

Measurements

Time (ms)
0.99707
1.00088
0.99611
0.9973
0.99587
0.99683
0.98324
1.01995
0.9973
0.9985
0.99659
0.98634
0.99635
1.02925
1.01233
0.99683
1.01757
0.99707
0.9985

Table J.1: The measurements of the REST API server request delay in milliseconds

Bibliography

[1] Agile Business Consortium. Moscow method, https://www.agilebusiness.org/content/moscow-prioritisation, 2014. Retrieved May 6, 2019.

[2] Mohiuddin Ahmed, Abdun Naser Mahmood, and Jiankun Hu. A survey of network anomaly detection techniques, 1 2016. ISSN 10958592.

[3] Bernhard Amann, Matthias Vallentin, Seth Hall, and Robin Sommer. Extracting certificates from live traffic: A near real-time ssl notary service. *Technical Report TR-12-014*, 2012.

[4] Zheng Dong, Kevin Kane, and L. Jean Camp. Detection of Rogue Certificates from Trusted Certificate Authorities Using Deep Neural Networks. *ACM Transactions on Privacy and Security*, 19(2):1–31, 9 2016. ISSN 24712566. doi: 10.1145/2975591.

[5] Zakir Durumeric, James Kasten, Michael Bailey, and J Alex Halderman. Analysis of the https certificate ecosystem. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 291–304. ACM, 2013.

[6] Kevin P Dyer, Scott E Coull, Thomas Ristenpart, and Thomas Shrimpton. Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. In *2012 IEEE symposium on security and privacy*, pages 332–346. IEEE, 2012.

[7] Gilberto Fernandes, Joel J.P.C. Rodrigues, Luiz Fernando Carvalho, Jalal F. Al-Muhtadi, and Mario Lemes Proença. A comprehensive survey on network anomaly detection, 3 2019. ISSN 15729451.

[8] Joseph Gardiner, Marco Cova, and Shishir Nagaraja. Command & Control: Understanding, Denying and Detecting - A review of malware C2 techniques, detection and defences. *arXiv preprint arXiv:1408.1136*, 8 2014. URL http://arxiv.org/abs/1408.1136.

[9] Ian Goodfellow. *Deep learning*. The MIT Press, Cambridge, Massachusetts, 2016. ISBN 9780262035613.

[10] Crane Hassold. A quarter of phishing attacks are now hosted on https domains: Why?, Dec 2017. URL https://info.phishlabs.com/blog/quarter-phishing-attacks-hosted-https-domains.

[11] intermax certificaties. Onze certificeringen: voldoen aan de strengste normen, 2019. URL https://www.intermax.nl/cloudsourcing/certificeringen/.

[12] ISO. Information technology — Security techniques — Information security management systems — Requirements. Standard, International Organization for Standardization, Geneva, CH, March 2013.

[13] Gareth James. *An introduction to statistical learning : with applications in R*. Springer, New York, NY, 2013. ISBN 978-1461471370.

[14] Seong S. Kim and A. L.N. Reddy. Statistical techniques for detecting traffic anomalies through packet header data. *IEEE/ACM Transactions on Networking*, 16(3):562–575, 6 2008. ISSN 10636692. doi: 10.1109/TNET.2007.902685.

[15] Rikard Laxhammar. Anomaly Detection. In *Conformal Prediction for Reliable Machine Learning: Theory, Adaptations and Applications*, pages 71–97. Elsevier Inc., 2014. ISBN 9780123985378. doi: 10.1016/B978-0-12-398537-8.00004-3.

[16] Kingsly Leung and Christopher Leckie. Unsupervised anomaly detection in network intrusion detection using clusters. In *Proceedings of the 38th Australasian Computer Science Conference*, volume 38, pages 333–342, Australia, 2005. Australian Computer Society, Inc. ISBN 1920682201.

[17] Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. Identifying suspicious urls: an application of large-scale online learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 681–688. ACM, 2009.

[18] Samuel Marchal, Jérôme François, Radu State, and Thomas Engel. Phishstorm: Detecting phishing with streaming analytics. *IEEE Transactions on Network and Service Management*, 11(4):458–471, 2014.

[19] Samuel Marchal, Kalle Saari, Nidhi Singh, and N Asokan. Know your phish: Novel techniques for detecting phishing sites and their targets. In *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, pages 323–333. IEEE, 2016.

[20] Rami M Mohammad, Fadi Thabtah, and Lee McCluskey. Predicting phishing websites based on self-structuring neural network. *Neural Computing and Applications*, 25(2): 443–458, 2014.

[21] NEN. Informatiebeveiliging in de zorg. Standard, Nederlands Normalisatie-instituut, Delft, NL, December 2017.

[22] Omid Omidvar. *Neural Networks and Pattern Recognition*, page 298. Academic Press, nov 1997. ISBN 9780125264204. URL https://www.xarg.org/ref/a/0125264208/.

[23] Charlie Osborne. Socs shift to threat detection and response: Gartner, mar 2019. URL https://www.zdnet.com/article/socs-shift-to-threat-detection-and-response-gartner/.

[24] Takaya Saito and Marc Rehmsmeier. The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. *PLOS ONE*, 10 (3):e0118432, March 2015. doi: 10.1371/journal.pone.0118432. URL https://doi.org/10.1371/journal.pone.0118432.

[25] Ivan Torroledo, Luis David Camacho, and Alejandro Correa Bahnsen. Hunting Malicious TLS Certificates with Deep Neural Networks. In *Proceedings of the 11th ACM Workshop on Artificial Intelligence and Security*, pages 64–73. ACM, Association for Computing Machinery (ACM), 10 2018. doi: 10.1145/3270101.3270105.

[26] Asrul H. Yaacob, Ian K.T. Tan, Su Fong Chien, and Hon Khi Tan. ARIMA based network anomaly detection. In *2nd International Conference on Communication Software and Networks, ICCSN 2010*, pages 205–209, 2010. ISBN 9780769539614. doi: 10.1109/ICCSN.2010.55.