Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelor Thesis

Cornelia Brülhart

Analysis of available Data Mining Algorithms to detect Advanced Persistent Threats (APT)

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Cornelia Brülhart

Analysis of available Data Mining Algorithms to detect Advanced Persistent Threats (APT)

Bachelor Thesis eingereicht im Rahmen der Bachelor thesis

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Klaus-Peter Kossakowski
Zweitgutachter: Prof. Dr.-Ing. Olaf Zukunft

Eingereicht am: 9. November 2016

Cornelia Brülhart

Thema der Arbeit

Analysis of available Data Mining Algorithms to detect Advanced Persistent Threats (APT)

Stichworte

Data Mining, Advanced Persistent Threat, Algorithmus, Entscheidungsbaum, Klassifikation, Weka, Netzverkehr Analyse, Perl, Netzwerk Angrifferkennungssystem, Überwachtes Lernen

Kurzzusammenfassung

Ziel dieser Bachlorarbeit ist das Entwickeln einer Methodenkette, welche Funktionen aus dem Bereich des Data Mining beinhaltet, um fortgeschrittene, andauernde Bedrohungen aufzudecken. Sowohl PDNS als auch NetFlow Logdateien werden hierbei mit einer Reihe an Perl Skripten transformiert und vorverarbeitet, um anschließend in einem Data Mining Programm (Weka) mit einem Algorithmus ausgewertet zu werden. Die Angriffs Detektion wird mithilfe eines Ampel-Konzeptes realisiert, welches IP Adressen nach ihrem Grad des Angriffsverhalten klassifiziert und in Listen speichert.

Cornelia Brülhart

Title of the paper

Analysis of available Data Mining Algorithms to detect Advanced Persistent Threats (APT)

Keywords

Data Mining, Advanced Persistent Threat, Algorithm, Decision Tree, Classification, Weka, Network Traffic Analysis, Perl, Network Intrusion Detection System, Supervised Learning

Abstract

The objective of this thesis is to determine a chain of different methods from the field of data mining, for the detection of advanced persistent threats. For this, both PDNS and NetFlow data log files are examined with multiple Perl scripts for preprocessing and data transformation purposes, as well as inserted into a data mining tool (Weka) to apply algorithms for knowledge discovery. To aid the detection of attacks, a traffic light concept for suspicious communication behaviour is being presented, which yields the automated developing of several lists containing IP addresses with varying levels of suspiciousness.

Contents

List of figures

Listings

1 Introduction

1.1 Background

Corporations deal with a significant quantity of diverse and often sensitive data and are therefore increasingly under attack by multiple types of hackers. Attacks can be executed by black hat hackers (being paid to illegally break into networks), hacktivists (politically or religiosly motivated to expose wrongdoing), state sponsored hackers (governments funding hacker groups) or spy hackers (hired by corporations to infiltrate and steal the competition‘s data). A breach in the companie‘s networks however is not only a business risk but involves also the customer‘s loss of trust in the company with regards to future transactions.
Current trends like Internet of Things, Cloud Technology and Industry 4.0 require particular attention in security as its goal is the interlinking of industrial practices with information technology, hence creating large amounts of data. With cyber attacks still on the rise, organizations try to contain threats with the use of information security teams, which apply firewalls, anti-malware software and even Intrusion Detection Systems (IDS). These components may have the ability to thwart some attacks but not the most sophisticated and inconspicuous ones.[3] [18]
Network intrusion where specific targets are selected to leverage advanced attack techniques are termed Advanced Persistent Threats (APTs). The attacker‘s goal is to remain undetected as long as possible while using various malware to acquire confidential information. There are currently several tools available for intrusion detection but none of them being able to detect reliably APTs due to the variety and complexity of the attacks.
Managing large data quantity to seek potential infiltration is a task, which can be realised with data mining (DM). Data mining involves the exercising of particular algorithms to extract patterns and meaningful knowledge from data.[14] The purpose of this thesis is thus to evaluate the application of multiple data mining algorithms as a method for APT attack detection.

1.2 Scope

The province of data mining contains many different algorithms and approaches, each suitable depending on the instance. The difficulty with finding appropriate algorithms for APT detection lies in the fact that attackers only have to trick one employee to open or release malware which leads to hidden vulnerability exploitation. With humans always being the weakest link in a network it is extremely challenging to prevent an APT attack before intrusion. Attacks can be targeting both executives as well as regular employees with e.g. spear-phishing or watering hole attacks, making the attack vector even more diverging. The highest success rates in APT defense lies therefore rather in attack detection than prevention. Data mining algorithms may help prevent or recognize attacks in many circumstances, but the challenge is to determine specifically which method is the most favourable.
The goal of this thesis is to develop a chain of procedures to learn the network's behaviour and determine a method of scanning for obtrusive IP addresses. The hypothesis is therefore that the application of selected, together stringed data mining tasks possible APT intrusion discloses. The aim is to develop a multistep technique to aid the process of seeking for traces of network invasion and unauthorized activity by analyzing large data sets of log files. This developed technique contains multiple steps since only applying anomaly detection algorithms on these vast data sets would take too much time and not yield the desired quality of recognition accuracy. This can be managed with the usage of preprocessing tasks prior to the appliance of algorithms, which will be further discussed in chapter 3.1.2.

1.3 Target audience

This thesis is intended for both technical staff of organization / national governmental CERTs and any in IT Security interested parties like security architects, network design engineers and technically-minded students. The primary spreading of malware can spring from any node in a network and therefore any user. Every organization, independently from its size, should instruct its employees about the importance of IT security and the correlation between their actions and potential intrusion. This document assumes, that the reader is familiar with the general network architectures and components as well as basic concepts of IP protocols and logging. Applied algorithms will be specifically elucidated or refered to its descriptive scientific paper in the footnote.

1.4 Outline

This thesis's structure is as follows: Chapter two contains the fundamentals and theory of data mining and knowledge discovery steps to establish a general understanding and reasoning for a specific path. Data mining is hereby being explained in the broader sense of the procedure of obtaining information from a set of data using knowledge discovery tasks. Specifically the task of preprocessing is of grand interest, as the application of algorithms can only yield usable results, if the data has been prepared for evaluation. Thereafter is chapter three, containing explanations of the selected approach and created scripts, as well as analysis of obtained results. Chapter three also includes the introduction of the traffic light concept for different level of suspiciousness and it's explanation. This is concluded with chapter four where achievements are being summarized and compared with the hypothesis and finished with the outlook section, where possible scenarios and functionalities are being listed.

2 Theory section

2.1 Network architecture

Before the analysis of potential attack vectors and development of an intrusion detection algorithm chain, a broad knowledge of typical network structures is essential. Figure 2.1 depicts a simplified configuration of a generic network architecture in corporations or governments. [24]

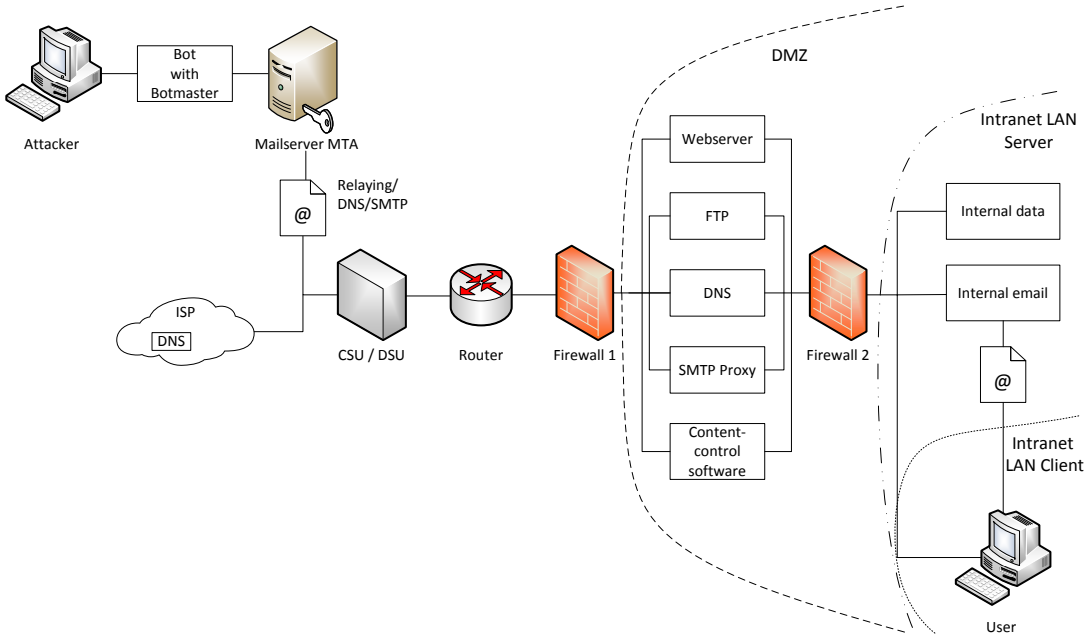

Figure 2.1: Exemplary Network Architecture

The corporation's interface to the internet is usually a router, which manages data packets between networks and has optionally a channel service unit / data service unit (CSU / DSU) before itself, connecting a terminal to a digital line and performing protective functions against electrical disturbances. The router is followed by a firewall (FW) or packet filter (here labeled Firewall 1 and also called "front-end"), applying filtering techniques and marking the

barrier to the corporation's demilitarized zone (DMZ). The DMZ acts as an additional interface between a trusted and an untrusted network segment, thus ensuring further partition and safety. Within the DMZ and following the firewalls are components like a webserver, FTP server and SMTP server. [22]
Each one of them is capable of generating log files in various formats with varying extent (e.g. webserver log files contain only a fraction of the full HTTP request / response). They are usually connected to a second firewall ("back-end"), isolating the DMZ from the intranet. However, imposing further restrictions with two firewalls provides additional security as well as increased administrative maintenance. The intranet with LAN can be divided into server and client party, with the user being on the client side. [21]

2.2 What is Data Mining?

Automated systems generate data for different purposes in large quantities. Examples are financial interactions (ATM machines), internet traffic, sensors or user interactions which lead to collecting data in exabyte ranges. However to gain useful information about the system and product the data has to be processed. [26]
Data mining is hereby the practice of automatically analysing data with the goal of extracting knowledge for further usage. After collecting data it has to be extracted, cleaned and analytically processed before commencing the return of a valuable feedback. The outcome can (depending on the objective) be either the prediction of possible / likely outcome or the description of data. The application of DM however involves several steps which are challenging to realize since even in related problems each data set may differ in format or type i.e. in quantitative, categorical or even graph-oriented values.
For simplification each data mining application is being connected to either clustering, classification, association pattern mining or outlier problems. Categorizing into these major building blocks helps to structure the process of optimal information retrieval from raw data sets. [33]

2.3 Data Mining in the Process of Knowledge Discovery in Databases

Acquiring specific information from a large data set is difficult and requires several steps which altogether can be called the Knowledge Discovery in Databases (KDD) Process. KDD describes hereby the general act of finding knowledge in data by applying various methods of whom DM the analytic part covers. Figure 2.2 illustrates the iterative and interactive procedure from the selection of dataset to the actual consolidation of knowledge. [7] [15]

1. Selection: After understanding the target and scope, the first step is the selection of a data set or subset based on the objectives. This results in the target data on which later DM will be applied and a model generated. Therefore if the set is too small or lacking the important attributes the resulting knowledge can be faulty.

2. Preprocessing: The target data might have invalid values like outlier, noise or missing entries. Noisy data is removed, missing values are filled with e.g. average ones or entirely deleted and format standards are implemented. This step has the goal to ensure data reliability and involves different strategies for the unwanted data. DM algorithms can already be applied on the corrupt value in this context.

3. Transformation: Not all data features are relevant and useful for the KDD process, hence this step. By applying transformation methods, such as feature selection or dimension reduction, variables can be reduced to receive only the relevant values for DM. This is typically performed with an ETL tool (Extract Transform Load), which enables automated extraction, transformation and loading of data as well as additional tasks like communication between distinct databases or reading of different file formats. This step is therefore rather project specific.

4. Data Mining: This step involves the applying of efficient and intelligent methods based on the objective to either describe or predict data. After selecting one of the main algorithm tasks (clustering, classification, association pattern mining or outlier problem) a suitable algorithm is to be selected and adapted on appropriate parameters.

5. Interpretation / Evaluation: Finally visualization and knowledge techniques are being used to present the discovered and evaluated knowledge to the user in an understandable manner. This acquired intelligence can now be utilized for action on other systems.

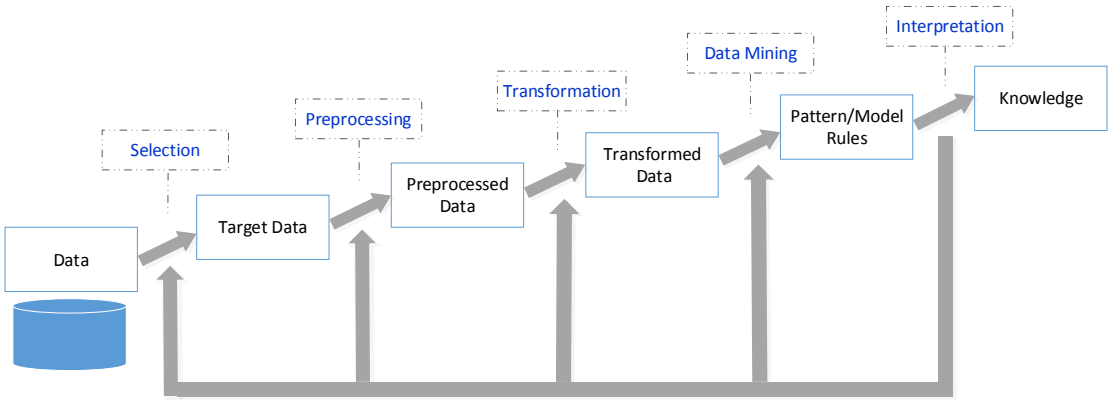

Figure 2.2: Data Mining in the Process of KDD

Each step has a feedback to the previous step to ensure iteration and loops between methods as well as interaction with the user for further modification.

2.4 Main Data Mining Tasks

As stated in chapter 2.2, each data mining task can be derived to one of the following four problems: Association pattern mining, outlier detection, clustering and classification. This is due to the fact, that they cover a wide range of possible scenarios, overlap and are related to one another.

2.4.1 Association Pattern Mining

Association Pattern Mining, today also often refered to as Association Rule mining, is a procedure with the objective of extracting relevant correlations and identifying frequent patterns or relationships between distinct attributes. A famous example of this method is in the context of items in a supermarkt: The supermarket owner might want to investigate if there is a dependency between certain products bought by his customers to adjust promotions or placement of products. If e.g. a customer buys beer it could be likely that diapers would be bought too, suggesting that dads are often told to do last-minute shopping, while the mothers are with the baby. This association rule would be expressed in the form X $\Rightarrow$ Y, with X and Y being the items and for this example as { Beer } $\Rightarrow$ { Diapers }.[1]

2.4.2 Outlier Detection

This task covers the detection of both outliers and anomalies.
Outliers are unusual data that differ greatly from the other observations (or median) and thus raise suspicion for faulty behavior. These anomalies can however contain meaningful information about unusual system characteristics which could lead to useful knowledge. They are nevertheless legitimate data points from valid observation, unlike anomalies, which are illegitimate data points that have been generated by a different process than the rest of the data. Anomaly detection is applied to uncover e.g. (network) intrusion, or fraud in various fields like insurance risk modeling, medical diagnosis or even law enforcement. Studying weather patterns to determining climate changes or natural disasters e.g. involves the collecting of large data sets. Anomalies and outliers in the data provide important insights about environmental trends and causes. [1][8]

2.4.3 Clustering

Clustering describes the process of partitioning objects into smaller groups based on their similarities. The creation of clusters allows a summarization and understanding of the data. A particular importance before applying clustering, lies in the first step of the KDD process, data selection, to reduce dimensionality in large data sets. This leads to better learning performances/accuracy and interpretability. The clustering methods can be categorized into density-based, partitioning or hierarchical methods. Shopping behaviour can be determined with clustering approaches, e.g. by comparing customer buying the same items and categorizing them. [33]

2.4.4 Classification

The last main DM task, classification allocates data points in a data set to a target class by applying a model. This model has been trained based on already partitioned training data from an example group and is referred to as the training model. The aim is to predict the target class of each unseen data point. Since the generating of a model requires an example data set to learn the structure of the classes, this task is refererred to as supervised learning. The necessity of a training model is crucial and the major difference to clustering. A possible scenario for classification might e.g. be the determination of loan viability in the banking industry. [19][33]

Data Mining can also be divided into six tasks, adding summarization and regression to the four depicted tasks. However as stated, the classes are related to one another and overlap

greatly in various algorithms. Summarization deals with the selection of a data subset to represent the properties and structure of the entire set. This is also found in association pattern mining.
Regression is used for prediction and has the aim to estimate the relationship and behaviour of data by creating a regression function. Data entries can be dependent and independent from another and might induce different behaviour depending on the scenario. Forecasting with regression might be regarded as a data classification method, but differs in the output. Regression has a continuous target value, predicting a response, whereas classification categorical target variable creates and therefore a class membership identifies.[10][4]

2.5 Advanced Persistent Threats

Ensuring information safety in organisations and governments has always been difficult and important but the emerging of Advanced Persistent Threats (APT) has shown that the sophistication of intrusion and caused damage is nowadays at a whole new level. Unlike other attacks this threat has the objective to access sensitive information stealthily from a specific target for as long as possible. While there are different definitions of the threat actor, the description of an APT can be well stated by summarizing the term components:[31] [14] [1]

Advanced: The threat actor (usually a group of hackers) has an adaptive mindset, possesses high technical skills and is usually well resourced. This enables compromising the target, gaining and maintaining access to networks by exploiting an organisation‘s weaknesses which can either be technical or human.

Persistent: The actor does not seek immediate financial gain but rather uses monitoring and hidden interactions over the course of years to obtain certain information steadily.

Threat: This term depicts, that the actor is a human with intent and motivation, not an automated piece of code, using different kinds of tools and backdoors. This usually also means, that the operator is well funded and organized.

2.5.1 Attack Methology / Kill Chain

The operators can be politically driven (espionage or even a directed attack) as well as targeting organizations in sectors with high-value information such as the financial- / mining- or oil industry. Some of the most famous attacks have e.g. been “Moonlight Maze“ which targeted

the computers at the Pentagon, NASA and research laboratories, “Operation Aurora“ where intellectual property of Google has been stolen and “Stuxnet“ where the stuxnet computer worm has been used to compromise Iran‘s nuclear program. However most companies / governments which have been targeted are either unaware of the (still ongoing) attack or just wish to not impart the situation to the public. The main concerns are the damage in reputation and thus profit loss as well as not wanting to provide information to the hackers about the system‘s point of breach.
The attack methodology, which can also be called “Kill Chain“ consists typically of the following six steps. Each one of them is interrelated with each other and describe the structure of intrusion. Disrupting any of the steps means that the operators effort is thwarted for certain time. [13] [31] [17]

1. Reconnaissance: This step is deemed one of the most critical steps for the actor since it is about gathering all kinds of information about the target. The success of an attack often depends on how thorough and extensive the gathering is executed. Used methods can be domain brute forcing, DNS cache snooping and web service exploration to retrieve technical data and search engine scripting, directory harvesting and spear phishing for nontechnical data.

2. Preparation: The attack is being prepared by testing particular tools or hardware (USB flash drives), identifying specific software versions, determining user name structure and vulnerabilities (e.g. zero-day expoits or social engineering).

3. Targeting: After identifying the vulnerabilities and informations in step 1. and 2. now the attack is being launched as well as monitored and the exploitation process initiated.

4. Further Access: To maintain access to the victim‘s resource, the operators quickly try to ensure a foothold in the compromised system by e.g. deploying backdoors to several machines (like using malicious device drivers that target the OS), installing rootkits and trying to obtain the administrator password. The type of backdoor using depends on the system and network.

5. Data gathering: The gathering manner of the relevant information depends on whether the victim has detected the attack or not. If detected, the operator will usually exfiltrate as quickly as possible to maximise yield. Otherwise they will chose the “slow and low“ methods to extract slowly (small quantities) and as unobtrusively as possible.

6. Maintenance: After successfully gathering information without detection the operators will clean up evidence of exploitation, preparation or even reconnaisance. To avoid detection the malicious activity is minimized and only periodic communication to the backdoors is implemented.

2.6 Algorithm Requirements and Procedure

The origin of malware distribution in a network can vary depending on the APT attack type and targeted vulnerabilities. APT intrusion itself may be conducted in a number of ways, with each attack being specifically customized for its target, making it even more hard to find traces in a vast set of network traffic.
However one succinct common manner of all APT attackers is the aim to stay undetected for as long as possible and gather data for a maximum period of time. Criteria to determine fit algorithms for the detection are therefore firstly, that the methods need to operate reliably on vast amounts of data, since each network component is capable of logging, and secondly the algorithm design should assure superior detection of inconspicuous occurences.
The strategy is to examine log files for attack vectors which means, that supervised learning methods are required, like support vector machine (SVM), linear regression or decision trees rather than clustering or anomaly detection. Furthermore, studies suggest, that supervised algorithms outperform unsupervised methods if the data contains no unknown attacks. Whereas their accuracy deteriorates significantly with a set containing unknown attacks. [12]
This thesis's procedure is as follows:
Network traffic data will be preprocessed and normalized for further examination, which means, scripts will be necessary to manipulate both windows and unix based traffic data. After preprocessing, the data is to be joined to create an instance containing relevant data from both traffic objects. This instance can then be used to classify IP addresses into three categories based on their behaviour with the internal network's components:

- **Very suspicious:** Displays harmful characteristics and is to be secured immediately. Entries with this feature can be pooled as a blacklist.
- **Suspicious:** These entries manifest some queer tendencies and can not be distinctively be categorized into a black- or whitelist and thus should be inspected further.
- **Harmless:** IP addresses declared as harmless are captured in a whitelist as they have not evoked any questionable conditions. There is no need to examine them promptly.

The task of classifying is to be performed with the conduction of decision trees, as they offer several characteristics which are advantageous for this task. Firstly, there is their hierarchical structure, which makes them easy to read and interpret as well as their ability to be rapidly adjusted and increased. For this thesis an exemplary count of three categories for IP address behaviour suffices, but with increased complexity the tree can easily be expanded.
Another advantage is the output construction of decision trees, as the aim is the generating of IP addresses lists, which are highly interpretable and can simply be extracted by exporting the nodes.

2.6.1 Weka

Realizing the task of data mining requires the application of automatic analysis on large data sets to extract interesting information. This task can be achieved with semi-automatic software applications ranging from open-source to proprietary tools. The selected workbench for this thesis is Weka (Waikato Environment for Knowledge Analysis), an open-source collection of machine learning algorithms. It has been developed by the University of Waikato in New Zealand in 1993 and is licensed under the General Public License (GNU) with the current stable release being 3.8.0 from April 14, 2016. It runs on Windows, OS X and Linux and can be called from Java code or used with the graphical user interface, the latter being the selected approach. [29]
Supported file formats include CSV, LibSVM, ARFF or C4.5 format, which can be loaded into Weka from multiple sources, such as databases or URLs. Its workbench includes algorithms of all four data mining subclasses, as mentioned in chapter 2.4 with additional visualization features and statistical evaluation. Weka has the advantages of simplicity and portability, since the GUI allows quick learning and it is advisable to use Weka with the aid of Java code with large data sets (> 10 GB). It is however not as fast as other tools e.g. R, which is written in C and Fortran. [9]

2.6.2 Decision Trees

Decision trees offer the possibility of arranging features into a tree-like structure, thus offering decision support for multiple domains e.g. calculating financial probabilities for cost-effectiveness of research and business decisions or classification of multi-leveled vision in the field of object recognition.[32]
A tree is built from the root to the leaf, by comparing objects and deciding, whether it matches the conditions of the path (input features), which are the internal nodes and can therefore be

placed in the leaf node. Internal nodes are termed split criteria as they are the conditions that the data has to fulfil to be passed along to the next criterion. For example, if a split criterion is $Height \leq 180\ cm$, the left branch of this internal node will contain all data with the height up to 180 cm, whereas the right branch will contain all entries greater than 180 cm. The split criteria goal is to maximize the division of the different leafes and can be done by univariate or multivariate splitting. [1]

Univariate splitting
This split handles the splitting of one variable and can be conducted on three types of attributes:

1. **Categorical attributes:**
   It is possible for categorical variables to take on one of r different values, with r being a limited and defined value. Applying a split can be performed with an r-way split, where every branch corresponds to one possible value of r. Another possibility that can be executed, if r is not large, is the practice of a binary split, which tests each and every one of the $2^r - 1$ combinations and selects the best one.

2. **Binary attributes:**
   These variables result in the generating of a binary tree, whose branches match one of the binary values.

3. **Numeric attributes:**
   Variables can also contain a fixed number r of ordered values within a small range. Splitting these can also be executed with a r-way split on each value or by using a binary condition and then selecting the best one.

Multivariate splitting
Multivariate splitting is performed, if nodes are to contain more than one attribute as a split criterion e.g.

$$Height \leq 180\ cm\ /\ Eyecolour = hazel$$

The resulting decision trees are much shallower, which indicates, that multivariate criteria are more powerful than univariate splitting criteria. Shallower trees are fundamentally more desirable, as the leafes contain larger data sets and are deemed to be less likely to overfit the model.
Data-dependant accuracy and overfitting are the most prominent problems with decision trees. Overfitting is the state in which a decision tree has been constructed to this level, that it fits the

data to well, thus even classifying noise variances, which can be avoided with the utilization of a pruning mechanism after tree construction to remove overfitting nodes.

2.6.3 Tree Evalutation

Weka provides several algorithms to generate decision trees: Hoeffding tree, LMT, M5P, random tree, REP tree or J48.

Hoeffding Tree: This algorithm is supported by the Hoeffding bound and categorized as a "Very Fast Decision Tree" (VFDT). It can be constructed incrementally and simultaneously with the data's arrival and offers the advantages of not requiring the entire data when building the tree and performing well with large data set. However drawbacks are the delayed node building and the static architecture of the tree, as nodes once built can not be changed or reversed. Since the data set is allocated as a whole and tree building is required to take place promptly, this algorithm is not deemed as the superior approach.

Logistic model trees (LMT): The LMT incorporates linear logistic regression and tree induction, thus offering the advantage of estimating probabilities while classifying. However, interpretability is aggravated as a consequence and externally calculated probabilities are not wished at this step, therefore is this approach not selected.[11]

M5P: This tree implements the M5 algorithm invented by R.Quinlan by recursively splitting the dataset based on the value of a chosen splitting attribute with the aim to minimize prediction error at each branch. M5 model trees offer the advantage of superior prediction accuracy and compactness over regression trees since they are able to extrapolate. They excel however when determining numeric prediction, which are not required for this thesis's set of problems.

Random Tree: Another by Weka offered classifying approach is the random tree, constructing a tree by randomly selecting attributes at each node and thus also not being the right algorithm for blacklist generating, since the tree is requested to base its branche building on the flags which represent the criteria for a possible attack.

Reduced Error Pruning (REP): REP trees are based on minimizing the error resulting from variance and being also categorized as a fast decision tree learner, however displaying inferior behaviour to J48. [34]

J48: The selected classifier is J48, since the desire for this definition of task is the building of a binary decision tree for IP address categorization. J48, an open source Java implementation of the C4.5 algorithm in Weka, is an extension to the ID3 algorithm. C4.5 generates a primary (unpruned) tree by pursuing the concept of divide-and-conquer and implementing rulesets. Starting at the root every branch becomes a prototype rule, whose instances are the consequences along the path. Each rule is being adjusted by analyzing the effect of rejecting every instance, with the aim to possibly discard an instance. C4.5's disadvantages are the large amount of required memory space and CPU time. It's advantages however overweigh: Simplicity, dealing with noise and easy-to-interpret model building and is therefore qualified to generate the desired lists.
There is an improvement of the C4.5, called C5.0 (since 1997), offering new capabilities and various enhancements like improved scalability of the decision trees and rulesets, boosting capability to optimizy predictive accuracy or permission for new data types. [16]

2.6.4 NetFlow

The term NetFlow depicts the service of gathering IP related network traffic with routers to understand network behaviour and is provided by Cisco. Applying services like NetFlow can help improve network security, since it facilitates the detection of e.g. unauthorized traffic, new applications‘ impact and even some anomaly detection by analyzing IP flow. The monitored IP packets consist of five to seven attributes: [30]

- IP source address
- IP destination address
- source port
- destination port
- layer three protocol type
- class of service
- router or switch interface

There are already many NetFlow versions (one to ten), with version five, seven and nine being the best documented ones and version five and nine (frequently used versions) having the most common export packet format types. A significant difference between NetFlow version five and version nine is, that version five uses a "static format" data structure, whose record

information cannot be extended to a non-version-five like format and function properly. Unlike version nine, which utilizes a dynamic template format of "data flowsets", allowing the adding of fields to NetFlow export packets.
The output of Netflow is a flow record, which in version nine consists of a packet header and one or more template / data FlowSets. FlowSets are collections of records (template FlowSets), which follow the packet header in an export packet or collections of data (data FlowSets), that have been grouped. If a flow is finished, the router will output the flow record. This data can be accessed with the Command Line Interface (CLI), providing an immediate view of the network situation or by exporting the data to a NetFlow collector (server), to assemble and analyze the information routinely. [5]

2.6.5 Nfdump

Nfdump consist of multiple tools for linux-based systems to process netflow data for traffic tracking or analysis. The input are flows which can be read in by a router and accumulated / stored by nfcapd (one of nfdump's tools), a netflow capture daemon. The output file is rotated every n minutes (n typically being 5) and stored as nfcapd.YYYYMMDDhhmm with the interval's time and date as the timestamp. Further nfcap handling is expounded in the application subsection. [23]

2.6.6 Passive DNS (PDNS)

Passive DNS is a system, which stores DNS record data for given location and time period to enhance DNS security. The domain name system (DNS) offers a distributed database for record sets, such as IP addresses and matching domain names on the Internet. It is however a very volatile infrastructure, which is constantly under attack, e.g. distributed denial-of-service attack (flooding the target with requests) or spoofing attacks (compromising the DNS server itself). Passive DNS captures inter-server DNS messages from authorative name servers and forwards them to a collection point of analysis in a time-stamped and compressed format. This recursive approach captures therefore anonymously server-to-server communication and can be applied with software, such as "dnstap". There are several different databases, to which sensors upload and collect data, which can then be reviewed for incident analysis. The concept has been originally developed by Florian Weimer in 2005 and presented at the FIRST conference.
One of the multiple uses of Passive DNS is the monitoring of changes to the resource record types A, AAAA (see chapter 3.1.2) over a span of time, indicating phishing, as legitimate domain names rarely change their address / name servers.

There are numerous RR types, with the following being relevant for this thesis: [25]

A / AAAA: The RR type A returns a 32-bit IPv4 address to map hostnames to the IP address of the host, whereas AAAA the same with 128-bit IPv6 addresses performs.

CNAME: These RRs create a synonymous name (canonical format) for the hostname, e.g. to hide network implementation details from clients.

MX: The MX (mail exchange) RR defines a mail exchange server to process or forward a mail for the domain name with the use of the Simple Mail Transfer Protocol (SMTP) to its destination mail server.

SOA: Every DNS zone contains a SOA RR at the beginning of the zone, containing authorative information about the zone (e.g. serial number or authorative server).

SRV: The SRV record specifies the service location of the desired servers and is often used instead of MX.

NF: RR typed NS indicate the authorative server zones, where every zone must contain as many as one NS recort at the root of the zone.

PTR: PTR is the counterpart to the A resource record, as it maps an IP address to a host.

2.6.7 Format

The raw data is a collection of log files from different sources, each having its distinct format and structure. Features like date or time can be represented in various ways (e.g. Aug. 5, 2016 or 2016-08-05) and need to be converted into a collective format, before the applying of any algorithm. Scripts or tools are therefore required, to recognizes the present data format and translate it into a mutual structure. [1]
The Internet Engineering Task Force (IETF) developed an object-oriented format called Intrusion Detection Message Exchange Format (IDMEF) for alert data, with exact regulations for characters, time formats and identifiers. IDS tools like PRELUDE, SNORT and SURICATA are able to generate data in IDMEF format (alerts) and generate Heartbeat messages, which are sent periodically to indicate current activity of the analyzers. [6]
The provided NetFlow data for this thesis is in ".xz" file format and can be extractet with nfdump, using a linux based system (here: Ubuntu 16.04.1), and then transformed to a CSV file

for further use. Since the file contains less relevant information for this investigation as well, a script is used (NFPreprocess.pl see appendix A.5) to create a CSV file with only significant data columns. Weka requires the CSV data to be in specific format or preferably in ARFF (Attribute-Relation File Format), to ensure control over the object types.
ARFF data are ASCII text files describing the structure and type of the data by creating a header and data section. The header contains the relation's name (@RELATION), list of attributes (@ATTRIBUTE) and their type (NUMERIC, NOMINAL, STRING or DATE <date-format>), whereas the data section (@DATA) the attribute's data contains.[28]

3 Method

3.1 Approach

There are already many IDSs on the market to aid with the detection of non-authorized network infiltration for both anomaly detection and misuse detection. The difference between both instances is, that anomaly detection involves the analysis of specified user behaviour and system‘s activity relating to historical data, whereas misuse detection is based on prior declared abnormal or normal behaviour. The common denominator of most IDSs is that they usually focus on only certain components of the correlation process and thereby not deliver the desired low false positives and nonrelevant positives rates.
To prevent this, the approach developed in this thesis, is to conduct extensive preprocessing and administer each received object with the use of scripts to obtain a high level of accuracy. In addition, the thesis introduces Weka to apply a classification algorithm to extract knowledge from log files. The proposed algorithm chain in this thesis shall be as follows (as shown in figure 3.1 below):
The raw data (for this thesis: PDNS and NF log files) will first be inserted into the preparation module, which conducts three tasks: Preprocessing, object fusion and setting flags. The outcome is manageable log data, which can now be passed to Weka, executing the decision tree building and list creating. The obtained blacklist and list of suspicious IPs (greylist), is then to be presented to a selected expert to evaluate its validity. IPs known to be non-malicious can be removed from the black- or grey list and inserted to the white list to speed up future network analysis. This chain of methods can be repeated periodically with new log files to filter and enrich the generated three lists in every cycle. The following are descriptions of the specific practices. [27]

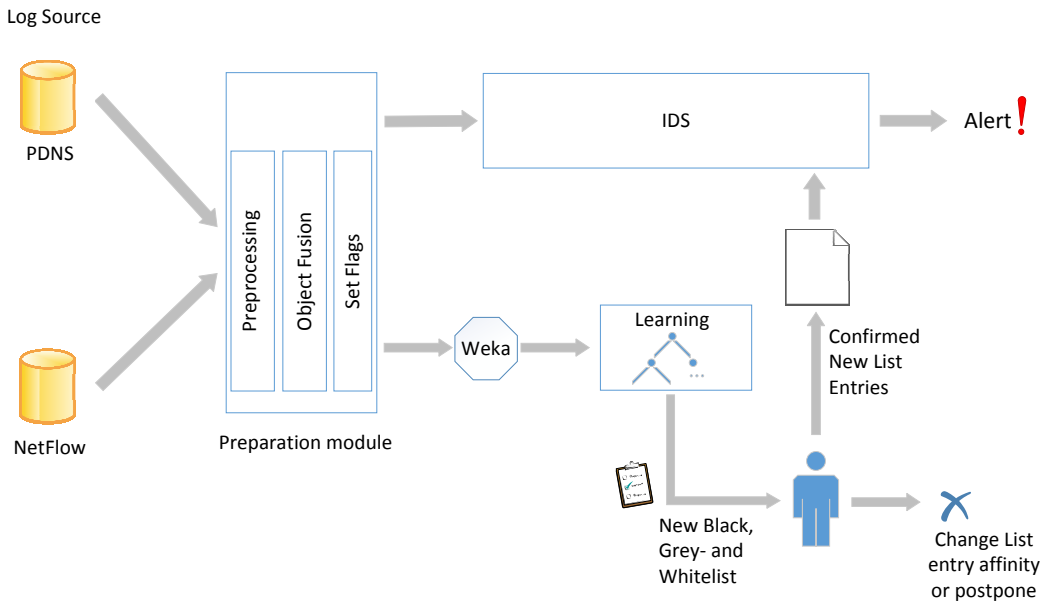

Figure 3.1: Method Chain

3.1.1 Selection

Before commencing with the selection of suitable data amounts, the log data has to be extractet and transformed to a comprehensible format. NF and PDNS logs are, as previously mentioned, in nfcapd- or pcap format stored, divided into more than thousand files and captured by a NF daemon or Wireshark. The first two generated scripts "pcapHandler.pl" (see listing A.3) and "nfcapHandler" (see listing A.1) in combination with "nfCSVmerge.pl" (see listingA.2) perform the converting to CSV. Each of these scripts is written in the programming language Perl, except for "nfcapHandler", which is a shell script. The functionality of "pcapHandler.pl" compared to "nfCSVmerge.pl" in combination with "nfcapHandler" is very similar, with the difference, that pcap files are being handled on a windows system and the NF data requires a Unix environment, hence the other two scripts.
"pcapHandler.pl"'s approach is firstly, to access the folder containing all pcap files and read the files's labels. This enables the automated running of a console command on every one of them, which is:

<path\tshark.exe> -tad -r <path\name of the current file> > <path\name of new file.csv>

The resulting CSV files (one CSV per pcap) are firstly being stored in a new folder and secondly being merged into one cumulative CSV file to simplify the task of data selection.
The Unix-based equivalent has slight differencies, as the approach is divided into conversion and the separate merging of all CSV files into one. Data selection can be performed easily on the two cumulative CSV files. Displaying the files is rather difficult due to data dimensions and can no longer be done with excel or notepad++, but is being realized with the tool UltraEdit, which is capable of handling large sets of CSV files.
Selecting data is a crucial step in KDD, as it creates the basis for following tasks and can determine the outcome of any knowledge discovery. The received log data is a network's traffic log data from a wednesday the 28.09.2016 2 AM to thursday 29.09.2016 2 AM and contains entries with varying protocols (UDP, TCP, ICMP for NF and DNS for PDNS). It would be the most interesting, to examine log files from several weeks and months, to really get acquainted with the employee's and network's characteristics. This way the task of learning the internal IP addresses for the "checkBehaviour.pl" (see listing A.7) script would be simplified and a rudimentary white list could also have been constructed. However, the available memory capacity is limited and even running all scripts on the whole CSV file containing 24 hours worth of logs would take too much time. The main bottleneck for calculations however, is Weka itself, as running large data invokes an "out of memory exception", thus not allowing the examination of large time intervals. With the given resources, Weka is just about able to calculate a 500 KB file containing over 20000 lines
To cover as many potential flag scenario as possible. the time 5:58 AM to 6:53 AM has been chosen, as the two minutes from 5:58 to 6:00 are within the irregular working hours range and thus will trigger the corresponding flag, perhaps as well as the other two (see chapter 3.1.4).

3.1.2 Preprocessing / Transformation

The data selection is succeeded by data manipulation in form of normalization with feature selection and is being executed by the script "PDNSPreprocess.pl" (see listing A.4) for PDNS and "NFPreprocess.pl" (see listing A.5) for NF data. The NF or PDNS CSV file is to be called as an argument when running any of both scripts.
Both preprocessing scripts exhibit at this point, why the choice of programing language for data manipulation scripts is Perl: It handles strings and characters efficiently with the usage of regular expressions, provide rapid application development due to its typelessness and is especially favourable when combining different and powerful components together. [20]
Many features known from other programing languages are also existent and if not, can often be incorporated with additional libraries (like Net::IP::Lite in "PDNSPreprocess.pl").

"PDNSPreprocess.pl" handles the DNS protocol entries and transforms PDNS data into the following format:

date (YYYY-MM-DD), time (hh-mm-ss), protocol, DNS-Server-IP (IPv4), query type, content1 (host), content2 (IP to host in IPv4)

Every entry is being dissected into log properties and stored for further enquire. DNS entries contain not only reverse look-up data about the given host but also resource records (RR) depending on the query request / response type. PDNSPreprocess.pl differentiates between the RR types: A, AAAA, CNAME, MX, SOA, SRV, NS and PTR, which are the RR types used in the provided data (for RR definitions see chapter 2.6.6).
Mail message transfers logs e.g. have the RR type MX and thus a different structure than RR in PTR type, which implement reverse DNS lookups, but return the IP in inverted order. Deviating entries are invalid and declared as zero in the content2 column to avoid empty fields. Besides rounding up the time field, further normalization is being conducted on some source, destination or host IPs responses which are in IPv6 and not in IPv4 format and are being converted into IPv4 format the the submethod "IPv6toIPv4". Both, the NF and PDNS, preprocess scripts also execute data transformation and select only necessary features and reduce the entry dimension, by leaving out additional flags and characters. However, PDNSPreprocess.pl aims to output only the log entries with the type "response", since these contain valuable information in the content2 field (IP addresses to the hosts, if available), and thus returning half of the raw data input (data without query requests).
NF data contains different information than PDNS logs, like port numbers or packets sent. The aim is to retrieve the following output format:

date (YYYY-MM-DD), time (hh-mm-ss), protocol, source IP (IPv4), source port number, destination IP (IPv4), destination port number, number packets sent, number bytes sent

NFPreprocess.pl (see listing A.5) realizes preprocessing by splitting each log entry and removing unnecessary blank spaces and arrows as well as rounding the time to whole seconds. In case of the protocol being ICMP, the destination port is being set as zero for echo replies. Transformation in form of feature reduction is being achieved by removing the entities "transaction duration" and "flow count".

3.1.3 Object Fusion

The last task of the preparation module is the fusion of the two preprocessed data objects, which represent the communication at the same section. This is performed by the "Connect-PDNStoNF.pl" (see listing A.6) script, which requires NF as the first and PDNS data as the second argument when executing. The desired output for this application is:

date (YYYY-MM-DD), time (hh-mm-ss), protocol, source IP (IPv4), source host (if found), number of source port, destination IP (IPv4), destination host (if found), number of destination port, number of bytes to destination, number of packets to destination, number of bytes to source, number of packets to source.

At first the preprocessed NF file is being checked for matching entries, where a request matches the response (with the antedecent entry being the outputted one). This is conducted by comparing source IPs with destination IPs and vice versa. When a pair is found, its' source or destination IPs are being compared to the response IPs from the PDNS data (content2), where a reverse look up has been successful, and if available the host information is appended. Conducting these tasks results in the combining of four log entries to one, containing only the important transaction information. Conducting this object fusion is furthermore part of the knowledge discovery.

3.1.4 Flags

Decision tree learning is a powerful data mining approach for various range of duties such as classifying objects by their nature. This thesis proposes a framework for an IP grading system to facilitate the detection of APT attacks. After having applied multiple tasks of preprocessing, the data can now be analyzed for the execution of the J48 decision tree in Weka. Each data entry is to be analyzed and marked for its level of suspiciousness with the script "checkBehaviour.pl" (see listing A.7). Entries are being screened for three indicators of an APT attack:

1. **Is there traffic beyond regular working hours?**
   The standard working hours for a medium-sized company with flex time is assumed to be from six o'clock in the morning to ten o'clock in the evening. Any irregular traffic is deemed suspicious and marked with a flag.

2. **Has there been data exchange with the exact byte size and the same addresses?**
   When finding multiple occurences of the same byte size exchange between two parties, it is to be examined first, in which direction the data transmission is taking place. The reason for this is to ascertain, if an internal component / network has been the victim of an APT attack and is sending data for two reasons:

   - **The infected object is confirming being still under the attacker's control**
     Attackers try to maintain power over networks / components for as long as possible, thus often not pursuing any data exfiltration for a long time after infection. To signalise the attacker an ongoing control over the network, infected devices periodically send a ping to a command and control server, which is called beaconing. This behaviour can ideally be detected with network data from a long period of time.
   - **The infected component is already transmitting data from the network**
     In case the attacker has already initiated the exfiltration and is trying to execute the thievery as unobtrusively as possible, he omitts large packages and sends more frequently.

3. **Is there traffic involving unpermissible ports?**
   Depending on the character of component types (client / server), among which an exchange takes place, some ports are considered undesirable and indicate intrusion. Setting a flag for suspicious port activity follows the illustrated concept in figure 3.2.

   Server offer both a server- and client port, to send or receive. It is important to define firstly which component is within the network or external for additional rule sets.
   Request initiator beyond the network can then be categorized into client and server:
   External clients sending requests to an internal component are only allowed to address a server on it's server port (smaller than 1023) and are not permitted to establish a connection to an internal client. Any instances violating these terms are considered very suspicious and marked with a flag.
   Server beyond the network may only send a request to an internal server from a client port (greater than 1023) to the internal server's server port. Connections to an interior client are prohibited.
   Internal clients on the other side may only address the internal server on the server ports. Any connection attempts from the client on the internal server's client port or any port on an external component is highly suspicious and marked.
   Server within the company's network are allowed to response to an external component

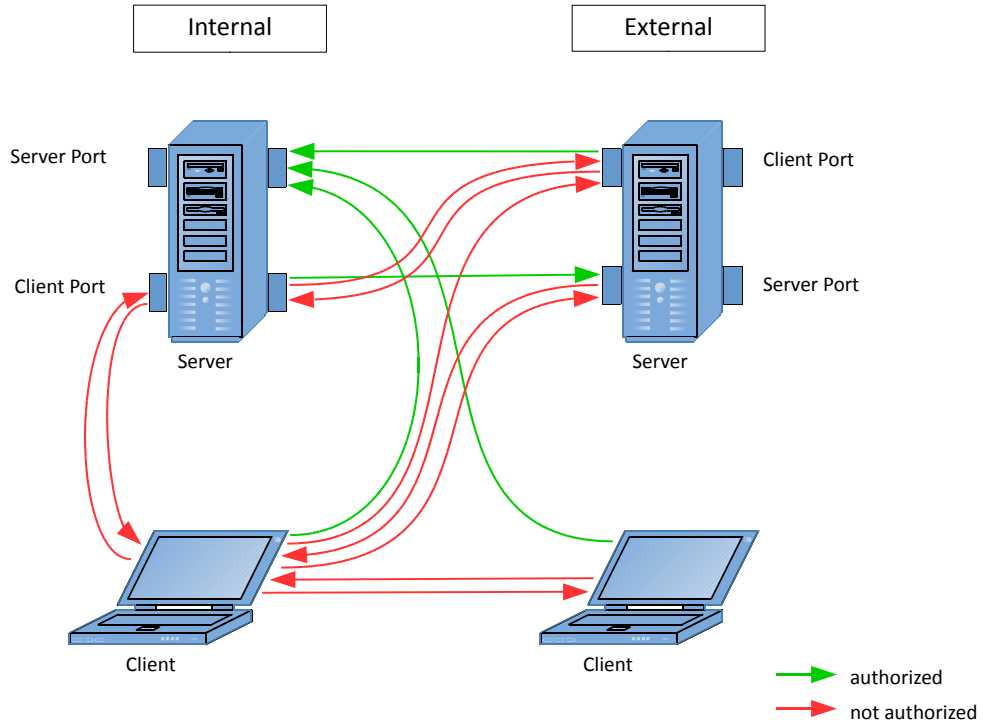

Figure 3.2: Client / Server Port Usage

from the server's client port. Responding an internal client is approved, if the responding server is using it's server port to reach the client.

The script "checkBehaviour.pl" performs the task of setting a flag if any of the above listed conditions applies. The specific approach is as follows:
To append the three flag columns to the preprocessed data, each entry is being extended by three zeros, representing the default setting of non-suspicious objects. Adjusting flags for traffic beyond working hours can be done by comparing the entry's timestamp to the approved working periods. Analysing the data for reocurring byte sizes however, requires comparing each entry to every following instance, multiplied by the number of the whole data. While iterating through data the first time, the script also generates a list of both internal server and client IPs to prepare instances for the subsequent flag setting in cases of unauthorized port usage. These lists are created through checking primarily, if the IP is within the network, by comparing their first three bytes (most significant byte is the first) to 249.210.50 or 249.210.51, which are known to be the first three bytes of the internal network.

Subsequently to determine, if the entry is a server or client, it is checked, if the port of the destination IP equals 22, 25, 53, 80, 123 or 443, which are the authorized server ports of an interior server, receiving a request. Actually setting any flags is performed in a second iteration of all entries, where any relations involving unsolicited ports can be examined with the use of generated interior server and clients lists.
Differentiating between internal servers and clients meets no difficulties, as a network administrator would have the IP addresses of the network's devices and components. The issues lies within the unknowing of components and intentions from outside, which always have the benefit of unlimited possibilities and time. Therefore is "checkBehaviour.pl" designed to cover as many malicious procedures involving the stated flag conditions as possible to ensure a strict black and greylist regime.

3.2 Application of Selected Algorithms

The perl script "checkBehaviour.pl" yields the entry format:

<IP address>,<flag for suspicious time>,<flag for repeated byte size>,<flag for suspicious port>

The decision tree can thereby be generated with the flags as the internal nodes to perform the classification of the given IP addresses. Before deploying the data in Weka, it might be desirable to convert the file into ARFF format. This can be realized with the Weka GUI Chooser, by selecting the ARFFViewer from the Tools selection, opening the CSV file and saving it as an ARFF document.
Some characteristics are already being displayed when opening the log file in Weka: The left-sided window contains the column identifiers and an attribute number, which is useful for further data manipulation in the Weka GUI. For example, attributes can be removed from the list, if deemed unnecessary for algorithm appliance, or different filter types can be selected in the filter menu at the top-center, which e.g. can convert numeric to nominal attributes. These filtering capabilities are very beneficial, if the task of preprocessing has not been extensive enough to conduct a Weka algorithm, since data can be quickly transformed into the desired structure.
The right-sided window contains detailed information of the selected attribute: Attribute details like name, type (nominal / numeric), count of distinguishable entries within this attribute or uniqueness (in percent) with regard to other instances. Numeric attributes are also briefly statistically analysed by determining a minimum, maximum and mean value of the selected

attribute, as well as a standard deviation from it. Nominal attributes are listed with a count and decimal number weight. A window with a brief visual overview of the selected attribute's distribution within the whole data set is right below the selected attribute information and can be adjusted to display the selected attribte distribution in relation to another attribute.
If all attributes are nominal, the classifying can be applied by selecting the Classify tab and choosing the classifier tree J48. However, to access the generated tree nodes and store them as lists, additional settings have to be adjusted in the Weka GUI Object Editor, especially the field "saveInstanceData" set to "true", to save each nodes content. Furthermore, the used class is the IP address and test options are set to cross-validation ten folds, as studies have suggested that a fold of ten retrieves the best estimate of error. [2]
Cross-validation is a technique to determine how accurately a model performs by partitioning data into sets. In this case the data is being partitioned into ten equal sized samples, declaring nine of them as training data to test the model and applying the model on the one remaining sample. This process is being executed ten times. Another validation technique, such as using a percentage of the data for validation (data partitioning) into two sets e.g. 20 % for testing and 80 % training are not suitable, as the whole data set is necessary for intrusion detection and relevant log data could be overlooked.

3.3 Outcome

Weka generates a decision tree (see fi gure 3.3) from the entered data with a height of three, where the root node is the flag condition "suspiciousPort", stating the remaining two flag conditions "suspiciousTime" and "repeatedByteSize" as internal nodes. All three flag occurences are portraying suspicious behavior, which could indicate APT activity. However they are not equally severe, which means the combination of the flag occurences is also relevant:
The root node "suspiciousPort" is the most critical factor, since unauthorized port activity could indicate command & control communication. IP addresses triggering this flag are most definitely of great interest and require analysis.
Either "suspiciousTime" and "repeatedByteSize" are conditions which express a similar low level of data suspiciousness as each one of them individually can feature actual legitimate connections. Communication logged at irregular working hours might suggest data exfiltration or botnet activity, but they can also be a operating system update, legitimate data transfer to other time zones or an employee working unusually late / early. Connections with repeated byte size occurences may be evaluated similarly, as hereby marked IP addresses portend beaconing characteristics or low-key piracy, but can also be coincidence or authorized automated email

checks. Suspicion for APT attacks can therefore be only raised with these two flags, if they appear in combination with any of the other two left conditions.
Listed IP addresses from the leafs can be either the client or server party, depending on the nature of behaviour. The generated tree allows the access to each node and performs no accuracy improvements, as the decision has been set (in the generic object editor) to "non collapsing" (no parts from the tree are being reduced that would not reduce training error), with a minimum number of object of one instance per leaf and no pruning performed, thus all nodes have been displayed. The decision tree itself can be displayed, by selecting "Visualize tree" in the result list. Its leafs are multiple lists of IP addresses with varying level of dangerousness, which are illustrated with reference to the traffic light colours, categorizing entries into lists (see figure 3.3):

- **Blacklist:** The red circled nodes in the left branch with the numbers six, seven and eight contain IP addresses, which have triggered all three flags and are highly suspicious (highlighted in red). Hosts can be identified with the aid of this list, which are either source or target of an attack. Further analysis could result in attack patterns which are composed of a sequence of individual attacks. Because of the J48 settings in the generic object editor, the tree is unpruned and not collapsed, therefore all nodes are being displayed to ensure detection of even small local features in the data set.
- **Greylist:** The yellow circled nodes with the number two, three, four and five are IP addresses causing one to two flags, hence display some questionable features. The flag count itself is not decisive, but rather the nature of the flag, as stated above.
- **Whitelist:** The remaining node number one, which is circled green does not have any marked flags and contain IP addresses, which are deemed harmless in this scenario and can be utilized for successive log examination.

Each leaf can be stored for further usage, by selecting the desired node and saving the node's content as an Arff file. When opening the Arff file node content in Weka, again a very helpful overview of the attributes is presented, but with the advantage that a distribution and count of the attributes reveals which IP addresses have been marked the most.

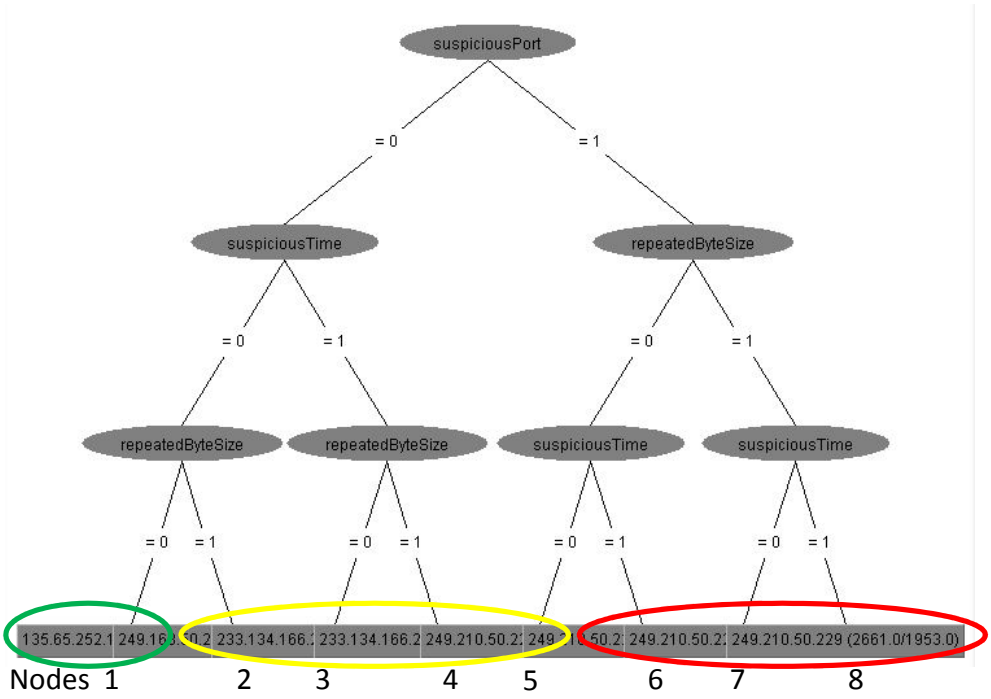

Figure 3.3: Generated Decision Tree

3.4 Analysis

Examination of the lists in Weka reveal the distribution and frequency of the IP addresses in its list. In general, it can be stated, that the most connections in the grey- and whitelist are reserved from the Internet Assigned Numbers Authority (IANA) with e.g. the IP address: 249.163.50.251. These IP addresses can now be categorized as non-harmful but have to be checked upon in the future.

The greylist contains also many occurences of connections to Apple Inc. with the IP address 17.199.49.146, which is one of the candidates, where the IT-expert would have to determine, if it is suspicious or not. In this case, the expert could transfer the IP address to the white list, as it would be extremely unlikely that an APT attack would originate from Apple.

The analysis of the blacklist has also revealed the highest occurence of connections established were to a reserved IANA entity, however the IT-expert should not research the highest number of connections but rather the unobtrusive calls, which have not emerged often.

The connection to the IP address 40.139.133.66 e.g. which is a supposed dental office in Aspen United States, would be an instance to check by the expert, since the connection occurence is low and the organisation unknown.
The examined data did not contain any real APT attacks and display only regular network characteristics, which explains, why no real malicious entities were found but rather recommendations made.

3.5 Coverage

APT detection can be attempted by several different approaches and this thesis's method is focused on the network traffic side. Multiple APT possibilities can be examined by concentrating on the log files of network components, such as unusual outbound network traffic, symptons of root kits or irregular access patterns.
The implemented chain of tasks has shown, that the preprocessing of data is an essential step, which determines the quality of the algorithm result. It is possible to extract knowledge from vast sets of data to study the network and determine possible APT attacks when examining network traffic. The selected classification algorithm, J48, generated a tree, on which a traffic light concept can be applied, to identify IP addresses with different levels of suspiciousness. The advantage of this, is its extensibility, as trees can be easily implemented on unidentified instances to study its nature. However some facts have also occured, which should be taken into consideration:
Weka is a very capable tool for data mining tasks but displayed some weaknesses, e.g. its long runtime when validating data or the problem of running out of memory, as the heap space fills quickly. This also led to the selection of the data to be only five minutes (see chapter 3.1.1). For future algorithm application it would be advisable to use Weka with its Java interface, which allows more resilient data manipulation.
Additionally, a disadvantage of perl is, that it is a interpretative language. This means, that it is comparatively slower to other languages like C. Its slowness has also been the reason, next to Weka's runtime, why the data selection resulted in the choice the 5 minute span of log files.

4 Discussion and Outlook

4.1 Summary

This thesis's goal was to create a concept with the help of data mining, which could facilitate the detection of APTs and possibly be connected to existing IDSs. To establish a common ground of knowledge, data mining has been explained in the conjunction of KDD, as finding vulnerabilities involves multiple KDD domains. Available data mining methods have been categorized into four major classes: Accociation pattern mining, clustering, outlier detection and classification. Each one of them offering unique approaches and possibilities for a broad province of tasks.
A goal was to determine which algorithm of theses classes could aid create an approach to reveal possible APT attack indicators. The selected approach is the systematic examination of network data to search for attack marks which in return could yield the IP address of the instance initiating this behaviour. Three exemplary attack marks have been suggested for this reason:

- traffic on suspicious ports
- action at unusual working hours
- and repeatedly same transmitted data sizes.

However before the selection of an algorithm, an extensive stage of preprocessing has been conducted, which is why this thesis required the creation of five different consecutive perl scripts, to ensure the reliability of data. Would the data have originated from one source, a preprocess script count of two would have sufficed. But since the intention was also to offer a perspective into realistic network interconnections, data collected from a unix and a windows based system has been employed. Merging both data sets has been an essential step to both understand each system's advantages and disadvantages as well as to learn possible obstacles when dealing with two very different and large data sets.
After having reduced and converted each data to a common denominator, both NF and PDNS data have been joined to create the essence of both information instances. A sixth perl script

has then been employed to perform marking of the three suspicious activity occurrences and preparing the data for following algorithm practice. Therefor the classification class has been selected with the specific choice of decision trees to conduct supervised learning on the received data. Different decision tree types have been evaluated, with the chosen tree instance being J48, an open source java implementation of the C4.5 algorithm in the data mining tool Weka. As a result of this, a multi-level tree has been generated, whose leaves are lists of IP addresses meeting the stated marking conditions to their degree. IP addresses fulfilling all three flag terms are classified as "very suspicious" and are listed in the blacklist of potential attackers, requiring definite review and action of an expert. Connections evoking only one to two of the flags are placed in the greylist, not needing immediate intervention, as many authorized connections might also invoke marking and thus require review. However other greylist entries could have the potential of being well-hidden attacks and should be checked for validity. Whitelist entries with no flags set are IP addresses categorized as reliable for future exchange.
Each generated list can be individually examined, processed in Weka for further dissection or implemented into IDS tools, offering valuable core information about network characteristics and potential attacks.

4.2 Conclusion

This thesis's main task has been the development of a chain of methods, to study a network's logging characteristics and determine an approach to identify suspicious IP addresses indicating an APT attack. This goal has been achieved with the creation of multiple perl scripts for different steps of preprocessing and the automated generating of black-, grey- and whitelists with the help of decision trees. The hypothesis, that a chain of data mining tasks possible APT detection discloses can be confirmed with the listed assumptions. APTs are highly sophisticated and complex attacks, which can be challenging to test or generalize, especially since they do not use commonly known attack signatures. However, the decision tree in this thesis is based on the conjecture, that APTs display some of the declared flag behaviours, which is an oversimplification. The hypothesis is therefore only confirmed for the given environment, as the decision tree would have to be improved with the addition of more flag conditions to aid available IDS in real life.
Also, the task of preprocessing is noted to be exceedingly vital for any attempts to gain information from vast log sets, which is why this thesis premised on the transformation and normalization of data. Every one of the three generated lists are in a CSV format and can

therefore be easily incorporated into available NIDS to improve intrusion detection and adjust filter rulesets of existing network guard components. The characteristic tree structure makes it especially advantageous, since any noticeable objects can be easily examined with a decision tree and partitioned into nodes of interest, thereby offering a valuable advancement to existent systems. Furthermore, an additional functionality has been implemented in the step of object fusion between NF and PDNS: The PDNS log files contain host names after the resource record type and when available, an IP address. If the NF data matches the PDNS log data with regards to time and place of recording, host names from the PDNS log files can be connected to the source or destination IPs from the NF data. This functionality occurs after the application of the "ConnectPDNStoNF.pl" script (see listing A.6), which inserts one column for the hostname after each the source and destination IP address. For this thesis however, the host names are not inserted (marking as zero, when not found), since the PDNS logs do not match the NF logs entirelly, due to the place of recording.

4.3 Outlook

The result of this thesis is a quantitative connection option for other NIDS with alert functionality due to its universality and uniform outputs. This offers multiple possibilities for further advancements. The most distinct one being the opportunity of connecting this thesis's concept to existing IDS. However the potential goes well beyond that:
Obtained lists can be returned into the chain for future network scanning to train the learning module and improve the detection accuracy. This could create a cycle of constant learning and advance even further when coupling or cross-checking results to trusted external lists e.g. Spamhaus or Distributed Sender Blackhole List, which are lists containing IP addresses from which large amounts of spam have been distributed.
Another extension could be linking the achieved chain to alert systems components, using the generated output to trigger explicit automated alerts which could be immediately sent to employees in a mail and intercept traffic. This method would shift the functionality from mere attack detection to attack prevention.
Furthermore, the whole potential of the generated method chain has not been fully reached as the host inscribing functionality is not being used. If the flagged data also contained host names, could the resulting lists not only include IP addresses but also names of organizations. This might be interesting for further alert notification and immediate filtering as mentioned above.

A Appendix

A.1 NF to CSV Converting

Listing A.1: nfcapHandler

```
#
#Script to convert NF data into CSV format
#
#Author: Cornelia Bruelhart
#SoSe 2016
#Hochschule fuer angewandte Wissenschaften Hamburg
#Bachelorthesis: Analysis of available Data Mining Algorithms to
    detect Advanced Persistent Threats (APT)
#

#!/bin/sh

counter=0
for file in /home/conny/Desktop/netflow-20160928/*;

do nfdump -r "$file" > /home/conny/Desktop/NFtoCSV/"nf"$counter.csv
counter=‘expr $counter + 1‘

done
```

A.2 NF CSV Merging

Listing A.2: nfCSVmerge.pl

```
#Script to merge all CSV NF files into one.
#
#Author: Cornelia Bruelhart
#SoSe 2016
#Hochschule fuer angewandte Wissenschaften Hamburg
#Bachelorthesis: Analysis of available Data Mining Algorithms to detect Advanced Persistent Threats (APT)
#
use 5.010;
use strict;
use warnings;

#Path of csv-directory
my $csvDirectory = 'Desktop\\NFtoCSV';

opendir(CSVFILE,$csvDirectory);
#Merge all csv-files to one

my @csvfiles = readdir(CSVFILE);
closedir(CSVFILE);

my $csvfilecounter = @csvfiles; #-3;
foreach(@csvfiles){

        if(($_ eq ".")||($_ eq "..")){
        next;
        }

        my $currentCSV = "Desktop\\NFtoCSV\\$_";

        open (my $data, '<', $currentCSV) or die "Could not open '$currentCSV' $!\n";
        while (my $line = <$data>){
        print $line;
        }
}
```

A.3 PCAP to CSV Converting

Listing A.3: pcapHandler.pl

```
#Script to convert all pcap files from a directory to csv-format and
    save in a folder named CSV, which already is on the desktop
#
#Author: Cornelia Bruelhart
#SoSe 2016
#Hochschule fuer angewandte Wissenschaften Hamburg
#Bachelorthesis: Analysis of available Data Mining Algorithms to
   detect Advanced Persistent Threats (APT)
#
use 5.010;
use strict;
use warnings;

#Path of the pcap- and csv-directory
my $pcapDirectory = 'Desktop\\passive-dns-20160928';
my $csvDirectory = 'Desktop\\CSV';

opendir(CSVFILE,$csvDirectory);
opendir(PCAPFILE,$pcapDirectory);
my @files = readdir(PCAPFILE);

#closedir(CSVFILE);
closedir(PCAPFILE);

my $counter = 0;

foreach(@files){
        if(($_ eq ".")||($_ eq "..")){
        next;
        }

$counter++;
my $cmd = "\"C:\\Program Files\\Wireshark\\tshark.exe\" -tad -r
   Desktop\\passive-dns-20160928\\$_ > $csvDirectory\\file$counter.
   csv";

```

```
33 system($cmd);
34 }
35
36 closedir(CSVFILE);
37 opendir(CSVFILE,$csvDirectory);
38 #Merge all csv-files to one
39
40 my @csvfiles = readdir(CSVFILE);
41 closedir(CSVFILE);
42
43 my $csvfilecounter = @csvfiles -3;
44 foreach(@csvfiles){
45         #print "blubb";
46         if(($_ eq ".")||($_ eq "..")){
47         next;
48         }
49
50         my $currentCSV = "Desktop\\CSV\\$_";
51
52         open (my $data, '<', $currentCSV) or die "Could not open '
               $currentCSV' $!\n";
53         while (my $line = <$data>){
54         print $line;
55         }
56 }
```

A.4 PDNS Preprocessing

Listing A.4: PDNSPreprocess.pl

```
1 #
2 #Script to normalize log files containing pdns-entries. This script
     checks for the most common DNS-types like AAAA, PTR or NS and
     performs normalization.
3 #Important data, such as sourceIP or IP address to given host is
     being saved, whereby only the response queries are returned.
4 #
5 #Author: Cornelia Bruelhart
6 #SoSe 2016
```

```
7  #Hochschule fuer angewandte Wissenschaften Hamburg
8  #Bachelorthesis: Analysis of available Data Mining Algorithms to
       detect Advanced Persistent Threats (APT)
9  #
10 use 5.010;
11 use Text::CSV;
12 use Net::IP::Lite;
13
14 #Open file
15 my $file = $ARGV[0] or die "Need␣to␣get␣CSV␣file␣on␣the␣command␣line
       \n";
16 open(my $data, '<', $file) or die "Could␣not␣open␣'$file'␣$!\n";
17
18 print "Date,Time,Protocol,DNS-Server-IP,Type,Content1,Content2\n";
19
20 while (<>) {
21
22     my($preQuery, $data) = split(/Standard/);
23     ($no,$date,$time,$srchost,$arrow,$dsthost,$proto, $len) = split(
           "␣",$preQuery);
24     $_ = $data;
25
26         #Some entries contain the sourceIP in IPv6 format, thus the
               calling of the submethod IPv6ToIPv4
27         if($srchost =~ /:/){
28         $srchost = IPv6ToIPv4($srchost);
29         }
30
31         if($dsthost =~ /:/){
32         $dsthost = IPv6ToIPv4($dsthost);
33         }
34
35         if ( m/^ query response 0x.... (.*)/ ) {
36
37                 #A & CNAME( which returns the canonical value of the
                       requested page)
38                 #No furter manipulation required
39                 $ip = $srchost;
40                 $type = "response";
41                 ($tmp1,$content,$tmp2,$content2) = split( "␣", $1 );
```

```
42
43                  #In case of server failure
44                  if($tmp1 eq "Server"){
45                  $content = "Server_failure";
46                  $content2 = 0;
47                  }
48
49                  #In case of format failure
50                  if($tmp1 eq "Format"){
51                  $content = "Format_failure";
52                  $content2 = 0;
53                  }
54
55                  if($tmp2 eq"A"){
56                  #Leave variables as they are
57                  }
58
59                  #AAAA: AAAA Records return the requested IP in IPv6
                        format, which is converted to IPv4 below.
60                  elsif( $tmp2 eq "AAAA"){
61                  $content2 = IPv6ToIPv4($content2);
62                  }
63
64                  #MX: Extract the response IP in case of PTR or MX,
                        which occurs in the logs after the A record
65                  elsif( $tmp2 eq "MX"){
66                          my $i= 0;
67                          my @ipv4 = split("_",$data);
68                          foreach(@ipv4){
69                                  if($ipv4[$i] eq "A"){
70                                          $i++;
71                                          last;
72                                  }
73                                  else{$i++;}
74                          }
75                  $content2 = $ipv4[$i];
76                  }
77
78                  # NS: Is the response of type NS, return 0
79                  elsif($tmp2 eq "NS"){
```

```
80                      $content2 = "0";
81              }
82
83              #PTR responses return the reversed IP address of the
                    (first part of the)arpa-address
84              if($tmp1 eq "PTR"){
85              my @ipReverse = split(/\./, $content);
86              $content2 = "$ipReverse[3].$ipReverse[2].$ipReverse
                    [1].$ipReverse[0]";
87              }
88
89              #The submethod NoMatchFunc is called, when there is
                    no such name in the entries.
90              if($tmp1 eq "No"){
91                      my($content, $content2) = NoMatchFunc($data)
                            ;
92              }
93
94              #Assigns the value 0, if the field is empty.
95              if(!defined($content2)){
96                      $content2 = "0";
97              }
98      }
99
100     elsif ( m/^ query 0x.... (.*)/ ) {
101
102             $ip = $dsthost;
103             $type = "request";
104             ($tmp1,$content,$tmp2) = split( "␣", $1 );
105             $content2 = "0";
106
107             }
108
109     # The following rounds the miliseconds to seconds. Time
            format now: hh:mm:ss
110     my @timetmp = split(":", $time);
111     $seconds = sprintf ("%.0f", $timetmp[2]);
112     if($seconds < 10){
113             $seconds = "0$seconds";
114     }
```

```
115         $time = "$timetmp[0]:$timetmp[1]:$seconds";
116
117   if($type eq "response"){
118         print "$date,$time,$proto,$ip,$type,$content,$content2\n";
119    }
120 }
121
122 #Submethod to return the requested address in content and Strint "No
        maching page found" in content2
123 sub NoMatchFunc{
124         my @iptmp = @_;
125         my $arraycontent = $iptmp[0];
126         my $i = 0;
127         my @param = split("_",$arraycontent);
128                 foreach(@param){
129                         if($param[$i] eq "A" || $param[$i] eq "PTR"
                                || $param[$i] eq "SRV" || $param[$i] eq "
                                SOA" || $param[$i] eq "AAAA"){
130                         $i++;
131                         last;
132                         }
133                         else{$i++;}
134                 }
135         $content = $param[$i];
136         $content2 = "No_matching_page_found";
137         return($content, $content2);
138 }
139
140 #Submethod to convert the refered IPv6 to IPv4
141 sub IPv6ToIPv4{
142         my @iptmp = @_;
143         my $arraycontent = $iptmp[0];
144
145         #Fills up the field, if the address contains ::
146         $arraycontent = ip_transform( $arraycontent, { lead_zeros =>
                1});
147
148         my @ipv6 =split(":", $arraycontent);
149
150         my $v6part1 = hex( $ipv6[6]);
```

```
151         my $v6part2 = hex( $ipv6[7]);
152         my $ip41 = scalar($v6part1>>8);
153         my $ip42=scalar($v6part1&0xff);
154         my $ip43=scalar($v6part2>>8);
155         my $ip44=scalar($v6part2&0xff);
156
157         $arraycontent = $ip41.".".$ip42.".".$ip43.".".$ip44;
158         return ($arraycontent);
159 }
```

A.5 NetFlow Preprocessing

Listing A.5: NFPreprocess.pl

```
1  #
2  #Script to normalize log files containing netflow data
3  #
4  #Author: Cornelia Bruelhart
5  #SoSe 2016
6  #Hochschule fuer angewandte Wissenschaften Hamburg
7  #Bachelorthesis: Analysis of available Data Mining Algorithms to
       detect Advanced Persistent Threats (APT)
8  #
9
10 use 5.010;
11
12 my $SourceDirectory = 'Desktop\\NFtoCSV';
13
14 opendir(CSVFILE,$SourceDirectory);
15
16 my @files = readdir(CSVFILE);
17
18 my $counter = 0;
19 my $flag = 0;
20
21 my $outputFile = "E:\\PreprocessedNF\\nfPreprocessedAndMerged.csv";
22
23 foreach(@files){
24
```

```
25          if(($_ eq ".")||($_ eq "..")){
26          next;
27          }
28
29  my $currentCSV ="Desktop\\NFtoCSV\\$_";
30
31  open (my $data, '<', $currentCSV) or die "Could not open $currentCSV
        \n";
32  open(OutFILE, '>>', $outputFile) or die "Could not open $outputFile\
        n";
33
34  my @lines;
35
36  my ($date,$time, $srcIP, $dstIP, $srcPort, $dstPort, $proto, $bytes,
         $packets);
37  my $buffer;
38
39  #Store file in array
40  while (<$data>) {
41          chomp;
42          push @lines, $_;
43  }
44
45  #remove the first line, which contains no log data
46  shift @lines;
47
48  my $stringtmp;
49
50  foreach $stringtmp (@lines){
51
52  my @tmp = split('\s+',$stringtmp);
53
54          if(($tmp[0] eq "Summary:")||($tmp[0] eq "Time")||($tmp[0] eq
                "Total")||($tmp[0] eq "Sys:")||($tmp[0] eq "Date")){
55          next;
56          }
57          $date = $tmp[0];
58          ($time,$buffer) = split("\\.",$tmp[1]);
59          $proto = $tmp[3];
60          ($srcIP,$srcPort) = split(":",$tmp[4]);
```

```
61          ($dstIP,$dstPort) = split(":",$tmp[6]);
62          $packets = $tmp[7];
63          $bytes = $tmp[8];
64
65          if ( $proto eq "ICMP" ) {
66                  $dstPort = 0;
67          }
68          if($flag == 0){
69          print OutFILE "Date,Time,Protocol,SrcIP,SrcPort,DstIP,
                DstPort,Bytes,Packets\n";
70          $flag = 1;
71          }
72          print OutFILE "$date,$time,$proto,$srcIP,$srcPort,$dstIP,
                $dstPort,$bytes,$packets\n";
73          }
74  }
75  close(CSVFILE);
76  close(OutFILE);
```

A.6 Connect PDNS to NetFlow

Listing A.6: ConnectPDNStoNF.pl

```
1  #Script to match PDNS to NF log entries by comparing response IP
       addresses to source and destination IPs
2  #
3  #Author: Cornelia Bruelhart
4  #SoSe 2016
5  #Hochschule fuer angewandte Wissenschaften Hamburg
6  #Bachelorthesis: Analysis of available Data Mining Algorithms to
       detect Advanced Persistent Threats (APT)
7  #
8  use 5.010;
9  use Text::CSV;
10
11 #Open files
12 #First file is Netflow data
13 my $file = $ARGV[0] or die "Need to get CSV file as the first
       parameter on the command line\n";
```

```
14 open(my $data, '<', $file) or die "Could not open '$file' $!\n";
15
16 #Second file is the pDNS data
17 my $file2 = $ARGV[1] or die "Need to get CSV file as the second
       parameter on the command line\n";
18 open(my $data2, '<', $file2) or die "Could not open '$file2' $!\n";
19
20 #Store first file in array. This contains the Netflow data.
21 while (<$data>) {
22         chomp;
23         push @linesFirstFile, $_;
24 }
25
26 #Store second file in array. This contains the pDNS responses
27 while (<$data2>) {
28         chomp;
29         push @linesSecondFile, $_;
30 }
31
32 print "Date,Time,Protocol,srcIP,srcHost,srcPort,dstIP,dstHost,
       dstPort,bytesToDstIP,packetsToDstIP,bytesToSrcIP,packetsToSrcIP\n
       ";
33
34 my $arraySizePDNS = @linesSecondFile;
35 my $arraySize = @linesFirstFile;
36 my @OneOfPair;
37 my $tmp=0;
38         for ( my $i=1; $i < $arraySize; $i++){
39
40         ($date,$time,$proto,$srcIP,$srcPort,$dstIP,$dstPort,
              $bytesToD,$packetsToD) = split (",",$linesFirstFile[$i]);
41
42                 #Iterate through the rest of the log file and
                       compare it to the above selected
43                 for (my $j = $i+1; $j < $arraySize; $j++){
44
45                 my ($date2,$time2,$proto2,$srcIP2,$srcPort2,$dstIP2,
                       $dstPort2,$bytesToC,$packetsToC) = split (",",
                       $linesFirstFile[$j]);
46
```

```
47                          if(($srcIP eq $dstIP2)  &&($dstIP eq $srcIP2
                                ) && ($dstPort == $srcPort2) && ($srcPort
                                 == $dstPort2) ){
48
49                                  my ($srcHost,$srcHost2,$dstHost,
                                        $dstHost2);
50
51                                  for (my $p = 1; $p < $arraySizePDNS;
                                         $p++){
52
53                                          my ($datePDNS,$timePDNS,
                                            $protoPDNS,$ipPDNS,
                                            $typePDNS,$contentPDNS,
                                            $content2PDNS) = split ("
                                            ,",$linesSecondFile[$p]);
54
55                                          if($content2PDNS eq "No_
                                            matching_page_found"){
56                                          last;
57                                          }
58
59                                          if($srcIP eq $content2PDNS){
60                                                  $srcHost =
                                                        $contentPDNS;
61                                                  }
62                                          elsif($dstIP eq
                                            $content2PDNS){
63                                                  $dstHost =
                                                        $contentPDNS;
64                                          }
65                                          elsif($srcIP2 eq
                                            $content2PDNS){
66                                                  $srcHost2 =
                                                        $contentPDNS;
67                                          }
68                                          elsif($dstIP2 eq
                                            $content2PDNS){
69                                                  $dstHost2 =
                                                        $contentPDNS;
70                                          }
```

```
71                                          if(!defined($srcHost)){
72                                                  $srcHost = 0
73                                          }
74
75                                          if(!defined($dstHost)){
76                                                  $dstHost = 0
77                                          }
78                                  }
79
80                                  $OneOfPair[$tmp] = "$date,$time,
                                        $proto,$srcIP,$srcHost,$srcPort,
                                        $dstIP,$dstHost,$dstPort,
                                        $bytesToD,$packetsToD,$bytesToC,
                                        $packetsToC\n";
81                                  $tmp++;
82                                  last;
83                          }
84                  }
85          }
86 print @OneOfPair;
```

A.7 Set Flags

Listing A.7: checkBehaviour.pl

```
1 #
2 #Script to apply rules on the input and check for suspicious
     behaviour and attributes. Potential threat indicator are marked
     as flags in each entry object. Three
3 #tags are being set/checked: Has there been data exchange with the
     exact byte size (with the same IPs), is there unsupervised
     traffic at an unusual time and is
4 #there traffic on uncommon ports (in regard to the protocols).
5 #
6 #Author: Cornelia Bruelhart
7 #SoSe 2016
8 #Hochschule fuer angewandte Wissenschaften Hamburg
9 #Bachelorthesis: Analysis of available Data Mining Algorithms to
     detect Advanced Persistent Threats (APT)
```

```
10 #
11 use 5.010;
12 use Text::CSV;
13 use Switch;
14
15 #Open file
16 #File contains of the combination of pdns and NF entries. These
       entries have been selected and modified by the select.pl script.
17 my $file = $ARGV[0] or die "Need to get CSV file as the first
       parameter on the command line\n";
18 open(my $data, '<', $file) or die "Could not open '$file' $!\n";
19
20
21 #Store file in array.
22 while (<$data>) {
23         chomp;
24         push @linesFile, $_;
25 }
26
27
28 #print "Date,Time,Protocol,srcIP,srcHost,srcPort,dstIP,dstHost,
       dstPort,bytesToDstIP,packetsToDstIP,bytesToSrcIP,packetsToSrcIP,
       suspiciousTime,repeatedByteSize,suspiciousPort\n";
29 print "IPaddress,suspiciousTime,repeatedByteSize,suspiciousPort\n";
30
31 my $arraySize = @linesFile;
32
33 #Create two lists for the IP addresses belonging to the internal
       servers or the internal clients
34 my @internalServers;
35 my @internalClients;
36 my @OneOfPair;
37 my @linesSecondArray;
38
39 my $suspiciousTime;
40 my $suspiciousTime2;
41 my $repeatedByteSize;
42 my $repeatedByteSize2;
43 my $suspiciousPort;
44 my $suspiciousPort2;
```

```
45
46 my $serverIPCounter = 0;
47
48 #Attach three flags to the entries set as false. In Perl 0 = false
       1=true (anything but 0)
49 for ( my $h = 1; $h < $arraySize; $h++){
50         ($date,$time,$proto,$srcIP,$srcHost,$srcPort,$dstIP,$dstHost
               ,$dstPort,$bytesToD,$packetsToD,$bytesToC,$packetsToC) =
               split(",",$linesFile[$h]);
51         $linesSecondArray[$h] = "$date,$time,$proto,$srcIP,$srcHost,
               $srcPort,$dstIP,$dstHost,$dstPort,$bytesToD,$packetsToD,
               $bytesToC,$packetsToC,0,0,0\n";
52 }
53 my $secondArraySize = @linesSecondArray;
54
55 for ( my $i=1; $i < $secondArraySize; $i++){
56
57 ($date,$time,$proto,$srcIP,$srcHost,$srcPort,$dstIP,$dstHost,
       $dstPort,$bytesToD,$packetsToD,$bytesToC,$packetsToC,
       $suspiciousTime,$repeatedByteSize,$suspiciousPort) = split (",",
       $linesSecondArray[$i]);
58
59         #Compare the first three bytes, if the IP address is
               internal or external
60         my($first,$second,$third,$fourth) = split("\\.",$dstIP);
61
62         #Check if IP is internal
63         if($first==249 && $second==210 && (($third ==51)||($third
               ==50))){
64
65         #Check if the IP belongs to an internal server and store the
                IP in internalServers array
66         switch($dstPort){
67                 case [22,80,25,53,123,443]{
68                 if(grep{$_ eq $dstIP} @internalServers){
69                         #do nothing, is already in array
70                 }
71                 else{
72                 $internalServers[$serverIPCounter]=$dstIP;
73                 $serverIPCounter++;
```

```
74                  }
75                  }
76          }
77          }
78
79          #Split the time into parts of hours,minutes and seconds, and
                set flag to true, if the traffic took place from
               23:00:00 to 05:59:00
80          my ($hours,$min,$sec) = split(":",$time);
81
82          if(($hours < 6) || ($hours > 22)){
83
84                  $suspiciousTime = 1;
85                  $linesSecondArray[$i] = "$date,$time,$proto,$srcIP,
                        $srcHost,$srcPort,$dstIP,$dstHost,$dstPort,
                        $bytesToD,$packetsToD,$bytesToC,$packetsToC,
                        $suspiciousTime,$repeatedByteSize,
                        $suspiciousPort\n";
86          }
87
88          if( $repeatedByteSize == 1){
89          #Continue with next iterator, since the flag has already
               been set
90          next;
91          }
92
93          #Iterate through the rest of the log file and compare it to
               the above selected
94          for (my $j = $i+1; $j < $secondArraySize; $j++){
95
96          my ($date2,$time2,$proto2,$srcIP2,$srcHost2,$srcPort2,
               $dstIP2,$dstHost2,$dstPort2,$bytesToD2,$packetsToD2,
               $bytesToC2,$packetsToC2,$suspiciousTime2,
               $repeatedByteSize2,$suspiciousPort2) = split (",",
               $linesSecondArray[$j]);
97
98
99                  # Check, if there has already been an exchange of
                       data with the same packet size
```

```
100                 # (same packets to D = possible exfiltration ->
                        inside IP sends data regularly)
101                 # (same packets to C = possible exfiltration ->
                        outside IP sends data regularly)
102                 if(($bytesToD2 == $bytesToD) && ($dstIP2 eq $dstIP))
                        {
103                         my $insideBool = IsInternal($dstIP);
104
105                         if(!$insideBool){
106
107                         $linesSecondArray[$i] = "$date,$time,$proto,
                                $srcIP,$srcHost,$srcPort,$dstIP,$dstHost,
                                $dstPort,$bytesToD,$packetsToD,$bytesToC,
                                $packetsToC,$suspiciousTime,1,
                                $suspiciousPort\n";
108                         $linesSecondArray[$j] = "$date2,$time2,
                                $proto2,$srcIP2,$srcHost2,$srcPort2,
                                $dstIP2,$dstHost2,$dstPort2,$bytesToD2,
                                $packetsToD2,$bytesToC2,$packetsToC2,
                                $suspiciousTime2,1,$suspiciousPort2\n";
109
110                         }
111                 }
112                 elsif(($bytesToC2 == $bytesToC) && ($srcIP2 eq
                        $srcIP)){
113                         my $insideBool = IsInternal($srcIP);
114
115                         if(!$insideBool){
116                         $linesSecondArray[$i] = "$date,$time,$proto,
                                $srcIP,$srcHost,$srcPort,$dstIP,$dstHost,
                                $dstPort,$bytesToD,$packetsToD,$bytesToC,
                                $packetsToC,$suspiciousTime,1,
                                $suspiciousPort\n";
117                         $linesSecondArray[$j] = "$date2,$time2,
                                $proto2,$srcIP2,$srcHost2,$srcPort2,
                                $dstIP2,$dstHost2,$dstPort2,$bytesToD2,
                                $packetsToD2,$bytesToC2,$packetsToC2,
                                $suspiciousTime2,1,$suspiciousPort2\n";
118                         }
119                 }
```

```
120                 else {
121                 # Otherwise do nothing & leave flag as 0
122                 }
123         }
124 }
125
126 #Fill the internalClients array with IPs, since internal servers are
        now known
127 for ( my $k = 1; $k < $secondArraySize; $k++){
128         ($date,$time,$proto,$srcIP,$srcHost,$srcPort,$dstIP,$dstHost
              ,$dstPort,$bytesToD,$packetsToD,$bytesToC,$packetsToC,
              $suspiciousTime,$repeatedByteSize,$suspiciousPort) =
              split(",",$linesSecondArray[$k]);
129
130         #Compare the first three bytes, if the IP address is
              internal or external
131         my $srcBool = IsInternal($srcIP);
132         my $dstBool = IsInternal($dstIP);
133
134         #Check if srcIP is internal
135         if($srcBool){
136
137                 #Check if it is already declared as a internal
                      server- otherwise it‘s an internal client
138                     if (grep{$_ eq $srcIP}@internalServers){
139                     #do nothing, is already marked as a internal
                          server
140                     }
141                     else{
142                     push (@internalClients,$srcIP)
143                     }
144         }
145 }
146
147 #Set flag, if suspicious connection has been etablished / suspicious
        port used
148 for ( my $l = 1; $l < $secondArraySize; $l++){
149         ($date,$time,$proto,$srcIP,$srcHost,$srcPort,$dstIP,$dstHost
              ,$dstPort,$bytesToD,$packetsToD,$bytesToC,$packetsToC,
```

```
            $suspiciousTime,$repeatedByteSize,$suspiciousPort) =
            split(",",$linesSecondArray[$l]);
150
151         my $srcBool = IsInternal($srcIP);
152
153         #Check if srcIP is internal
154         if($srcBool){
155                 #Is it a internal server
156                 if(grep{$_ eq $srcIP}@internalServers){
157
158                         my $bool=IsInternal($dstIP);
159                         if($bool){
160                         switch($srcPort){
161                         case [22,80,25,53,123,443]{
162                                 my $clientBool = IsInternalClient(
                                        $dstIP);
163                                 if(($dstPort > 1023)&&($clientBool))
                                        {
164                                 #$linesSecondArray[$l] = "$date,
                                        $time,$proto,$srcIP,$srcHost,
                                        $srcPort,$dstIP,$dstHost,$dstPort
                                        ,$bytesToD,$packetsToD,$bytesToC,
                                        $packetsToC,$suspiciousTime,
                                        $repeatedByteSize,0\n";
165                                 $linesSecondArray[$l] = "$srcIP,
                                        $suspiciousTime,$repeatedByteSize
                                        ,0\n";
166                                 }
167
168                                 else{
169                                 #$linesSecondArray[$l] = "$date,
                                        $time,$proto,$srcIP,$srcHost,
                                        $srcPort,$dstIP,$dstHost,$dstPort
                                        ,$bytesToD,$packetsToD,$bytesToC,
                                        $packetsToC,$suspiciousTime,
                                        $repeatedByteSize,1\n";
170                                 $linesSecondArray[$l] = "$srcIP,
                                        $suspiciousTime,$repeatedByteSize
                                        ,1\n";
171                                 }
```

```
172                         }
173                         else{
174                                 my $serverBool = IsInternalServer(
                                        $dstIP);
175
176                                 if(($srcPort > 1023) && ($serverBool
                                        )){
177                                 #$linesSecondArray[$l] = "$date,
                                        $time,$proto,$srcIP,$srcHost,
                                        $srcPort,$dstIP,$dstHost,$dstPort
                                        ,$bytesToD,$packetsToD,$bytesToC,
                                        $packetsToC,$suspiciousTime,
                                        $repeatedByteSize,0\n";
178                                 $linesSecondArray[$l] = "$srcIP,
                                        $suspiciousTime,$repeatedByteSize
                                        ,0\n";
179                                 }
180                                 else{
181                                 #$linesSecondArray[$l] = "$date,
                                        $time,$proto,$srcIP,$srcHost,
                                        $srcPort,$dstIP,$dstHost,$dstPort
                                        ,$bytesToD,$packetsToD,$bytesToC,
                                        $packetsToC,$suspiciousTime,
                                        $repeatedByteSize,1\n";
182                                 $linesSecondArray[$l] = "$srcIP,
                                        $suspiciousTime,$repeatedByteSize
                                        ,1\n";
183                                 }
184                         }
185                         }
186                         }
187                         else{
188                         if($srcPort > 1023){
189                                 switch($dstPort){
190                                 case[22,80,53,25,123,443]{
191                                 #$linesSecondArray[$l] = "$date,
                                        $time,$proto,$srcIP,$srcHost,
                                        $srcPort,$dstIP,$dstHost,$dstPort
                                        ,$bytesToD,$packetsToD,$bytesToC,
```

```
                                    $packetsToC,$suspiciousTime,
                                    $repeatedByteSize,0\n";
192                                 $linesSecondArray[$l] = "$srcIP,
                                    $suspiciousTime,$repeatedByteSize
                                    ,0\n";
193                                 }
194                                 else{
195                                 #$linesSecondArray[$l] = "$date,
                                    $time,$proto,$srcIP,$srcHost,
                                    $srcPort,$dstIP,$dstHost,$dstPort
                                    ,$bytesToD,$packetsToD,$bytesToC,
                                    $packetsToC,$suspiciousTime,
                                    $repeatedByteSize,1\n";
196                                 $linesSecondArray[$l] = "$srcIP,
                                    $suspiciousTime,$repeatedByteSize
                                    ,1\n";
197                                 }
198                                 }
199                         }
200                         else {
201                         #$linesSecondArray[$l] = "$date,$time,$proto
                            ,$srcIP,$srcHost,$srcPort,$dstIP,$dstHost
                            ,$dstPort,$bytesToD,$packetsToD,$bytesToC
                            ,$packetsToC,$suspiciousTime,
                            $repeatedByteSize,1\n";
202                         $linesSecondArray[$l] = "$srcIP,
                            $suspiciousTime,$repeatedByteSize,1\n";
203                         }
204                         }
205                 }
206                 #Therefore it is an internal client
207                 else{
208                         my $bool=IsInternal($dstIP);
209                         if($bool){
210                                 my $serverBool = IsInternalServer(
                                    $dstIP);
211                                 if($serverBool){
212                                 switch($dstPort){
213                                         case [22,80,25,53,123,443]{
```

```
214                                     #$linesSecondArray[$l] = "
                                            $date,$time,$proto,$srcIP
                                            ,$srcHost,$srcPort,$dstIP
                                            ,$dstHost,$dstPort,
                                            $bytesToD,$packetsToD,
                                            $bytesToC,$packetsToC,
                                            $suspiciousTime,
                                            $repeatedByteSize,0\n";
215                                     $linesSecondArray[$l] = "
                                            $srcIP,$suspiciousTime,
                                            $repeatedByteSize,0\n";
216                                     }
217                                     else{
218                                     #$linesSecondArray[$l] = "
                                            $date,$time,$proto,$srcIP
                                            ,$srcHost,$srcPort,$dstIP
                                            ,$dstHost,$dstPort,
                                            $bytesToD,$packetsToD,
                                            $bytesToC,$packetsToC,
                                            $suspiciousTime,
                                            $repeatedByteSize,1\n";
219                                     $linesSecondArray[$l] = "
                                            $srcIP,$suspiciousTime,
                                            $repeatedByteSize,1\n";
220                                     }
221                                 }
222                                 }
223                                 else{
224                                 #$linesSecondArray[$l] = "$date,
                                        $time,$proto,$srcIP,$srcHost,
                                        $srcPort,$dstIP,$dstHost,$dstPort
                                        ,$bytesToD,$packetsToD,$bytesToC,
                                        $packetsToC,$suspiciousTime,
                                        $repeatedByteSize,1\n";
225                                 $linesSecondArray[$l] = "$srcIP,
                                        $suspiciousTime,$repeatedByteSize
                                        ,1\n";
226                                 }
227                             }
228                             else{
```

```
229                         #$linesSecondArray[$l] = "$date,$time,$proto
                                ,$srcIP,$srcHost,$srcPort,$dstIP,$dstHost
                                ,$dstPort,$bytesToD,$packetsToD,$bytesToC
                                ,$packetsToC,$suspiciousTime,
                                $repeatedByteSize,1\n";
230                         $linesSecondArray[$l] = "$srcIP,
                                $suspiciousTime,$repeatedByteSize,1\n";
231                         }
232                         }
233                 }
234
235         #SrcIP has to be external
236         else{
237         my $serverBool = IsInternalServer($dstIP);
238
239         if($serverBool){
240                 if($srcPort > 1023){
241                 switch($dstPort){
242                         case [22,80,25,53,443,1194]{
243                         #$linesSecondArray[$l] = "$date,$time,$proto
                                ,$srcIP,$srcHost,$srcPort,$dstIP,$dstHost
                                ,$dstPort,$bytesToD,$packetsToD,$bytesToC
                                ,$packetsToC,$suspiciousTime,
                                $repeatedByteSize,0\n";
244                         $linesSecondArray[$l] = "$srcIP,
                                $suspiciousTime,$repeatedByteSize,0\n";
245                         }
246                         else{
247                         #$linesSecondArray[$l] = "$date,$time,$proto
                                ,$srcIP,$srcHost,$srcPort,$dstIP,$dstHost
                                ,$dstPort,$bytesToD,$packetsToD,$bytesToC
                                ,$packetsToC,$suspiciousTime,
                                $repeatedByteSize,1\n";
248                         $linesSecondArray[$l] = "$srcIP,
                                $suspiciousTime,$repeatedByteSize,1\n";
249                         }
250                 }
251                 }
252                 else{
```

```
253                 #$linesSecondArray[$l] = "$date,$time,$proto,$srcIP,
                        $srcHost,$srcPort,$dstIP,$dstHost,$dstPort,
                        $bytesToD,$packetsToD,$bytesToC,$packetsToC,
                        $suspiciousTime,$repeatedByteSize,1\n";
254                 $linesSecondArray[$l] = "$srcIP,$suspiciousTime,
                        $repeatedByteSize,1\n";
255                 }
256         }
257         #An external component tries to connect to an internal
                client
258         else{
259         #$linesSecondArray[$l] = "$date,$time,$proto,$srcIP,$srcHost
                ,$srcPort,$dstIP,$dstHost,$dstPort,$bytesToD,$packetsToD,
                $bytesToC,$packetsToC,$suspiciousTime,$repeatedByteSize
                ,1\n";
260         $linesSecondArray[$l] = "$srcIP,$suspiciousTime,
                $repeatedByteSize,1\n";
261         }
262         }
263 }
264
265 print @linesSecondArray;
266
267 #Check, if the given IP is listed as an internal client
268 sub IsInternalClient{
269                 my @iptmp = @_;
270                 if (grep{$_ eq $iptmp[0]}@internalClients){
271                         return(1);
272                 }
273                 else{
274                         return(0);
275                 }
276 }
277
278 #Check, if the given IP is listed as an internal server
279 sub IsInternalServer{
280                 my @iptmp = @_;
281                 if (grep{$_ eq $iptmp[0]}@internalServers){
282                         return(1);
283                 }
```

```
284                 else{
285                         return(0);
286                 }
287 }
288
289 #Check, if the given IP is listed as an internal component
290 sub IsInternal{
291         my @iptmp = @_;
292         my($first,$second,$third,$fourth) = split("\\.",$iptmp[0]);
293         if($first==249 && $second==210 && (($third ==51)||($third
                ==50))){
294                 return(1);
295         }
296         else{
297                 return(0);
298         }
299 }
```

Bibliography

[1] Aggarwal, Charu C.: *Data Mining - The Textbook*. Springer, 2015. – URL http://dx.doi.org/10.1007/978-3-319-14142-8. – ISBN 978-3-319-14141-1

[2] Bengio, Yoshua ; Grandvalet, Yves: No Unbiased Estimator of the Variance of K-Fold Cross-Validation. In: *The Journal of Machine Learning Research* 5 (2004), Dec., S. 1089–1105. – URL http://jmlr.csail.mit.edu/papers/volume5/grandvalet04a/grandvalet04a.pdf. – access date: 2016-08-02. – ISSN 1532-4435

[3] Brecht, Daniel: *Current Trends in the APT World*. 2015. – URL http://resources.infosecinstitute.com/current-trends-apt-world/. – access date: 2016-10-15

[4] Cai, Eric: *Machine Learning Lesson of the Day - Supervised Learning: Classification and Regression*. 2014. – URL https://chemicalstatistician.wordpress.com/2014/01/05/machine-learning-lesson-of-the-day-classification-and-regression/. – access date: 2016-07-26

[5] Cisco: *Cisco IOS® NetFlow Version 9 Flow-Record Format*. 2011. – URL http://www.cisco.com/en/US/technologies/tk648/tk362/technologies_white_paper09186a00800a3db9.html. – access date: 2016-05-01

[6] Debar, H. ; Curry, D. ; Feinstein, B.: *The Intrusion Detection Message Exchange Format(IDMEF)*. 2007. – URL https://tools.ietf.org/html/rfc4765. – access date: 2016-09-01

[7] Fayyad, Usama ; Piatetsky-Shapiro, Gregory ; Smyth, Padharic: From Data Mining to Knowledge Discovery in Databases. In: *Fall 1996* 17 (1996), S. 37–54. – URL http://www.aaai.org/ojs/index.php/aimagazine/article/view/1230/1131. – access date: 2016-07-26

[8] Gerardnico: *Data Mining - (Anomaly | outlier) Detection*. 2001. – URL http://gerardnico.com/wiki/data_mining/anomaly_detection#outliers_vs_anomaly. – access date: 2016-05-11

[9] Hall, Mark ; Frank, Eibe ; Holmes, Geoffrey ; Pfahringer, Bernhard ; Reutemann, Peter ; Witten, Ian H.: The WEKA data mining software: an update. In: *SIGKDD Explorations Newsletter* 11 (2009), S. 10 – 18. – URL http://www.kdd.org/exploration_files/p2V11n1.pdf. – ISSN 1931-0145

[10] Jeevan, Manu: *Fundamental methods of Data Science: Classification, Regression and Similarity.* 2015. – http://www.kdnuggets.com/2015/01/fundamental-methods-data-science-classification-regression-similarity-matching.html

[11] Landwehr, Niels ; Hall, Mark ; Frank, Eibe: Logistic Model Trees. In: *Machine Learning* 59 (2005), May, S. 161–205. – URL http://www.cs.waikato.ac.nz/ml/publications/2005/LMT.pdf. – access date: 2016-08-11. – ISSN 0885-6125

[12] Laskov, Pavel ; Düssel, Patrick ; Rieck, Konrad: Learning intrusion detection: supervised of unsupervised? In: *Image Analysis and Processing - ICIAP 2005* 3617 (2005), S. 50 – 57. – URL http://link.springer.com/chapter/10.1007%2F11553595_6. – ISSN 0302-9743

[13] Ltd, Command Five P.: *Advanced Persistent Threats: A Decade in Review.* 6 2011. – URL www.commandfive.com/papers/C5_APT_ADecadeInReview.pdf. – access date: 2016-07-26

[14] Ltd, Command Five P.: *Command and Control in the Fifth Domain.* 2 2012. – URL www.commandfive.com/papers/C5_APT_C2InTheFifthDomain.pdf. – access date: 2016-07-26

[15] Marsh, Jennifer: *Knowledge Discovery in Databases: 9 Steps to Success.* 2001. – URL https://blog.udemy.com/knowledge-discovery-in-databases/. – access date: 2016-07-26

[16] Melese, Francois ; Richter, Anke ; Solomon, Binyam: *Military Cost-Benefit Analysis, Theory and practice.* 1st. Routledge Studies in Defence and Peace Economics, 2015. – ISBN 978-1-138-85042-2 , 978-1-315-72469-0

[17] Miller, Russell: *Advanced Persistent Threats: Verteidigung von Innen gegen Angriffe von Außen.* 7 2012. – URL www.ca.com/de/%7e/media/Files/whitepapers/ca-advanced-persistent-threats-wp-deu.pdf. – access date: 2016-07-26. – CA Technologies, Security Management

[18] Molok, Nurul Nuha Abdul M. ; Chang, Shanton ; Ahmad, Atif: Information Leakage through Online Social Networking: Opening the Doorway for Advanced Persistence Threats. In: *Journal of the Australian Institute of Professional Intelligence Officers* 19 (2011),

S. 38–55. – URL http://ro.ecu.edu.au/cgi/viewcontent.cgi?article=1092&context=ism. – ISSN 1039-1525

[19] Oracle®: *Oracle® Data Mining*. 2001. – URL https://docs.oracle.com/cd/B28359_01/datamine.111/b28129/classify.htm. – access date: 2016-07-26

[20] O'Reilly, Tim: *The Importance of Perl*. 2016. – URL http://archive.oreilly.com/pub/a/oreilly/perl/news/importance_0498.html. – access date: 2016-10-14

[21] Sharma, Ganesh D.: *Packet Filtering Firewall: An Introduction*. 2010. – URL http://securityworld.worldiswelcome.com/packet-filtering-firewall-an-introduction. – access date: 2016-08-17

[22] Shinder, Deb: *SolutionBase: Strengthen network defenses by using a DMZ*. 2005. – URL http://www.techrepublic.com/article/solutionbase-strengthen-network-defenses-by-using-a-dmz. – access date: 2016-08-10

[23] Sourceforge.net: *NFDUMP*. 2014. – URL http://nfdump.sourceforge.net/. – access date: 2016-08-13

[24] Stoddard, Donald ; Thomas, Thomas M.: *Network Security First-Step: Firewalls*. 2012. – URL http://www.ciscopress.com/articles/article.asp?p=1823359&seqNum=5. – access date: 2016-08-10

[25] TechNet, Microsoft: *Resource Record Types*. 2016. – URL https://technet.microsoft.com/en-us/library/cc958958.aspx. – access date: 2016-09-7

[26] Thearling, Kurt: *An Introduction to Data Mining*. 2012. – URL http://www.thearling.com/text/dmwhite/dmwhite.htm. – access date: 2016-07-26

[27] Valeur, Fredrik ; Vigna, Giovanni ; Kruegel, Christopher ; Kemmerer, Richard A.: A Comprehensive Approach to Intrusion Detection Alert Correlation. In: *IEEE Transactions on dependable and secure Computing* 1 (2004), S. 1 – 12. – URL https://de.scribd.com/document/88267766/A-Comprehensive-Approach-to-Intrusion-Detection-Alert-Correlation

[28] Vinterbo, Staal A.: *ARFF (developer version)*. 2016. – URL http://weka.wikispaces.com/ARFF+%28developer+version%29. – access date: 2016-11-05

[29] Waikato, Machine Learning G. at the University of: *Data Mining Software in Java*. 2000. – URL http://www.cs.waikato.ac.nz/ml/weka/index.html. – access date: 2016-08-02

[30] Whitepapers, Cisco: *Introduction to Cisco IOS NetFlow*. 2012. – URL http://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.html. – access date: 2016-06-01

[31] Wrightson, Tyler: *Advanced Persistent Threat Hacking: The Art and Science of Hacking Any Organization*. 1st. McGraw-Hill Education Group, 2014. – ISBN 0071828362, 9780071828369

[32] Wu, Xindong ; Kumar, Vipin ; Ross Quinlan, J. ; Ghosh, Joydeep ; Yang, Qiang ; Motoda, Hiroshi ; McLachlan, Geoffrey J. ; Ng, Angus ; Liu, Bing ; Yu, Philip S. ; Zhou, Zhi-Hua ; Steinbach, Michael ; Hand, David J. ; Steinberg, Dan: Top 10 Algorithms in Data Mining. In: *Knowl. Inf. Syst.* 14 (2007), no. 1, S. 1–37. – URL http://www.cs.uvm.edu/~icdm/algorithms/index.shtml. – access date: 2016-07-26. – ISSN 0219-1377

[33] Zhao, Qiankun ; Bhowmick, Sourav S.: Association Rule Mining: A Survey. no. 2003116, S. 1–20. – URL https://www.lri.fr/%7eantoine/Courses/Master-ISI/Regle-association.pdf. – access date: 2016-07-26

[34] Zhao, Yongheng ; Zhang, Yanxia: Comparison of decision tree methods for finding active objects. In: *Advances in Space Reasearch* 41 (2008), S. 1955–1959. – URL http://www.sciencedirect.com/science/article/pii/S027311770700796X. – ISSN 0273-1177

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 9. November 2016 Cornelia Brülhart