ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze

Nicolas Känzig

Network Monitoring and Attack Detection

Master Thesis MA-2018-02
March to September 2018

Tutors: Roland Meier[1], Dr. Luca Gambazzi[2], Dr. Vincent Lenders[2]
Supervisor: Prof. Laurent Vanbever[1]

[1]ETH Zurich, Switzerland
[2]armasuisse W+T, Thun, Switzerland

Abstract

Due to the rapid increase of sophisticated attacks on computer systems and networks, the field of network anomaly and intrusion detection has become a center of intense research during the past decades. The analysis of network traffic on a large scale and identifying suspicious patterns in the data has proven to be a major challenge.

In this thesis we investigate methods for performing effective analysis of network traffic and evaluate machine-learning based techniques for automated detection of malicious activities. We perform the analysis on a large set of raw network data originating from past cyber defense exercises (*Locked Shields*).

First, we apply common tools for network traffic analysis and intrusion detection such as *Wireshark*, *Bro* and *Snort* to the data. We then use the different traffic representations extracted by these tools to build up an extensive database (*Elasticsearch*), enabling powerful ways for analysis and visualization of the data. In addition, we label connections between compromised hosts and Command and Control (C&C) servers that are under control of the attacker team, using information sources provided by the organizers of the Locked Shields exercise.

In the second part of the thesis, we investigate possible machine-learning based applications for intrusion and anomaly detection to provide the defenders with an additional monitoring tool during the exercise. We train supervised machine-learning models that can predict sessions established to malicious C&C servers with high precision. We then conduct a thorough analysis of the model robustness by simulating adversarial inputs and packet loss, while proposing possible methods to increase the model's resilience. Based on these results, we propose an intrusion detection system that could greatly assist the analysts in detecting malicious activity during future competitions.

Finally, we evaluate unsupervised clustering approaches, first on the novel intrustion detection dataset CICIDS2017, where the malicious traffic is completely labelled, which facilitates the validation of the developed models. Going back to the Locked Shields data we proof the feasibility of unsupervised methods for intrusion detection on this data by reporting high detection rates of the C&C sessions mentioned above.

Acknowledgements

Foremost, I want to thank Professor Laurent Vanbever for giving me the opportunity to write this thesis at the Networked Systems Group. His provided insight and expertise greatly assisted the research.
I would like to express my gratitude to Roland Meier and Luca Gambazzi for their guidance. The discussions in our weekly meetings were always of great inspiration to me and were key to the success of this thesis.
Special thanks also Vincent Lenders for the interesting discussions, the critical feedback and for sharing his extensive knowledge in cyber-security.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Analysing and understanding network activity can be extremely challenging in large and heterogeneous networks. In case of security incidents, response teams often invest a significant part of their resources to understand the compromised network architecture and to identify suspicious activity pattern for subsequent investigations. The need rises for developing sophisticated systems allowing response teams to focus more on the incident resolution instead of consuming most resources on the analysis.

One of the main challenges is to effectively analyse and visualize the network traffic and to extract meaningful information from the data. Tools such as *Wireshark* [87] and *Bro* [67] enable network traffic analysis on a packet- and flow-level basis. Being able to look into every single packet and all established connections that went over the wire is crucial for each further analysis. However, manual inspection of the extracted information and detecting suspicious patterns in the data becomes increasingly difficult for network administrators and security analysts when dealing with big amounts of network traffic. On one hand this calls for ways to visualize and to extract meaningful statistics from the data to provide a more abstract view and making the information easier accessible for humans.
On the other hand, many researches over the past decades have focused on the development of automated and intelligent tools in the anomaly and intrusion detection domain [12]. Many of these investigations aim to develop machine-learning based techniques to identify malicious activities in network traffic data (see Section 1.2.2). The use of such algorithms is promising as they have proven to outperform humans in a variety of different pattern recognition tasks. However, due to the constantly changing attack profiles used by hackers and the variety of different deployed networks architectures, the development of such systems has proven to be an extremely challenging task [86].

1.1 Contributions

Throughout this thesis we investigate both, large scale network data analysis as well as the development of machine-learning based IDS solutions. In the following we summarize the main contributions we make in this thesis:

- We develop a large scale network data analysis framework, based on an extensive Elasticsearch database, containing different representations (packets, flows, alerts) of the traffic capture data originating from the Locked Shields 2017 & 2018 cyber-defense exercises.
- We perform a thorough analysis of the Locked Shields data using the visualization tool Kibana running on top of the created Elasticsearch database.
- We label of the C&C sessions in the Locked Shield 2017 & 2018 network traffic recordings.
- We design an Intrusion detection System (IDS), based on supervised machine-learning algorithms, able to detect C&C sessions with high precision during future Locked Shields exercises.

- We show that the C&C classifiers achieve significantly higher performance when using the novel feature set proposed by [73] rather than the popular KDDcup99 attributes [30].
- We conduct an extensive analysis of the robustness of the IDS and its resilience to attacks and packet loss.
- We evaluate the attack detection performance on the novel CICIDS2017 intrusion detection dataset [73] using unsupervised clustering algorithms.
- We provide a proof of concept for the feasibility of unsupervised detection algorithms on the Locked Shields data by reporting high detection rates of the C&C sessions using a fully unsupervised set-up.

1.2 Related work

In the following we provide an overview of existing work in the areas of large scale network data analysis and network anomaly and intrusion detection.

1.2.1 Network data analysis

When investigating security incidents, at a certain point security analysts and response teams often have to dive into network traffic analysis. For intrusion detection and prevention as well as for network forensics it is crucial to have means of visualizing the network traffic and extracting meaningful information from the data. For large scale and high-speed networks, developing and deploying such systems can be a challenging task.

In the past there have been several attempts in this direction both for live traffic monitoring as well as to perform offline analysis of traffic captures. In the following we list three popular frameworks that are related to the approach we pursue in this thesis:

Moloch *Moloch* is an open source, large scale, full packet capturing, indexing, and database system. It provides methods for efficient recording of network traffic in PCAP format [85]. The framework contains a simple web interface which enables PCAP browsing, searching, and exporting. A flow-based representation of the traffic is extracted from the PCAP data and exposed to the web interface.

In this thesis we develop a similar set-up to perform network data-analysis. However, we use the highly customizable data visualization tool Kibana [23] for browsing the network traffic. This enables generating new custom visualizations more seamlessly.

Apache Metron *Apache Metron* [5] is an extensive cyber-security framework developed for real-time network anomaly detection to enable organizations to rapidly respond to them. It provides a mechanism to capture, store, and normalize any type of security telemetry at extremely high rates and can be deployed in a distributed fashion (*Apache Kafka* [3] & *Storm* [4]). The data is processed in real-time and enriched using secondary sources such as threat intelligence, geolocation, and DNS information. Metron provides a rich set of parsers for common security data sources (PCAP [85], Netflow [18], Bro [67], Snort [76], Fireeye [31], Sourcefire [19]) as well as a pluggable framework to add new custom parsers for new data sources. In the future, developers are planning to include next generation defense techniques such as anomaly detection and machine-learning algorithms that can be applied in real-time [5].

In our set-up we use some of the same data sources (PCAP, Bro, Snort). However, Metron focuses on real-time deployment in a distributed fashion while we pursue creating a more lightweight set-up to perform offline forensic network data analysis.

Security Onion *Security Onion* [61] is a free and open source Linux distribution designed for network security monitoring and intrusion detection tasks. It includes popular networking and security tools such as Snort, Suricata [82], Bro, OSSEC [63], Sguil [72], Squert [81] and NetworkMiner [57] while deploying a Elasticsearch database [23] where the information from these sources is gathered. Kibana is used to search, view, and interact with the data stored in the Elasticsearch indices.

The Security Onion project is likely to be the most similar solution to our approach. However, while the framework creates indexes for Bro and Snort data, the tool does not include the original PCAP data into the database. This makes sense, as Security Onion aims to enable real-time deployment. Online indexing of the raw packet data would consume too many resources. Also, to use the framework, the Security Onion Linux distribution has to be deployed. Our approach on the other hand is more lightweight and platform-independent.

1.2.2 Network anomaly and intrusion detection

In the following we describe some of the different approaches taken to solve the anomaly and intrusion detection problem. For each approach we list some of the projects and investigations we find particularly relevant.

Rule-based IDS approaches Rule-based IDS with the ability to perform real-time traffic analysis such as *Snort* [69] and *Suricata* [82] have been widely deployed in industry. When using these tools, fine-tuning of the implemented rules to the network to be protected as well as careful evaluation of the triggered alerts is paramount. Even so, the lack of contextual awareness of these systems and the continuously evolving nature of the network anomalies makes reliable automation and rapid threat assessment a challenging task. The rule-sets of these engines are constantly being updated and extended in order to keep up with the constantly changing and increasingly sophisticated methods used by attackers. Despite these tools being around for over a decade, there is still active research concerning effective deployment of such systems [88, 38, 64].

Supervised learning IDS approaches Models in the supervised-learning domain assume the availability of a training data set that entails labelled instances for normal as well as anomalous traffic samples. While some works only distinguish between these two classes, others separate the anomalies further into different attack categories. A typical approach is to train a predictive model that classifies previously unseen samples either as normal or anomalous.
However, two major issues usually arise in supervised detection. First, obtaining representative and accurate labels for the data is challenging and often requires the efforts of domain experts. Second, the count of anomalous samples is usually fairly low compared to the normal instances in the data, leading to highly unbalanced classification tasks. Some of the most common supervised models used in the intrusion detection domain are: Support Vector Machine (SVM) [37, 17], Decision Trees [91, 90], Artificial Neural Networks (ANN) [70, 36], Naive Bayes Classifiers [7, 28] and K-Nearest Neighbours (K-NN) [45, 47].

Semi-supervised learning IDS approaches Semi-supervised learning is a class of supervised learning tasks making use of both labelled and unlabelled data, where typically only a small fraction of the data is labelled. The goal is to understand how combining labelled and unlabelled instances may change the learning behaviour, and to design algorithms that take advantage of this combination [93]. In the intrusion detection domain, this techniques are very promising as acquiring labels for the data is often costly and obtaining a complete labelling is a challenging task. However, most of these techniques rely on strong assumptions about the data. Bad matching of the problem structure with the model assumption can even lead to a degradation in classifier performance compared with fully supervised approaches [92].
Generally there are two classes of problem definitions in the semi-supervised IDS domain. In the first class, labels for a small fraction of the anomalies in the data are provided, which can be used to train classification based anomaly detection models. In the second class only the normal traffic samples are labelled to train a baseline model for normal-operation traffic, which enables the detection of anomalies as patterns that deviate from this model. In [83] a non-parametric, semi-supervised learning algorithm is proposed, demonstrating dramatic performance improvements over supervised learning approaches as well as signature-based IDS in previously unseen malicious network traffic, while generating an order of magnitude fewer false alerts. [8] presents a novel fuzziness based semi-supervised learning approach by utilizing unlabelled samples assisted with supervised learning algorithms to improve the classifier's performance, where originally only 10% of the anomalies in the training set are labelled. Inspired by the deep-learning domain, [58] investigates a novel method for anomaly detection based on estimating the density in the compressed hidden-layer representation of autoencoders.

Unsupervised learning IDS approaches Fully unsupervised learning approaches do not rely on any previously acquired knowledge about the data. In other words, these algorithms do not require labelled samples for training. Supervised-learning based detection systems are highly effective to detect those anomalies that were labelled in the training set, but cannot defend the network against new unknown attacks. Unsupervised models on the contrary are able to autonomously detect anomalies and to characterize traffic deviating from normal operation. However, these methods usually make the implicit assumption that normal instances are far more frequent in the data than anomalies. When this assumption is not hold, such techniques might suffer from severely high false positive alert rates.
[46] uses a method based on K-means clustering algorithm combined with particle swarm optimization to overcome the inherent convergence issues of the algorithm. In [53] unsupervised detection is accomplished by means of robust clustering techniques, combining sub-space clustering with correlation analysis to blindly identify anomalies. An extensive survey on unsupervised approaches in networking and intrusion detection tasks is provided in [86].

We summarize that each class of network anomaly and intrusion detection methods and systems has unique strengths and weaknesses. The suitability of an anomaly detection technique depends on the nature of the problem attempted to address and on the available data. Hence, providing a single integrated and optimal solution to every problem may not be feasible [12].

1.3 Outline

The contents of this thesis are organized as follows.

In Chapter 2, we provide the background knowledge required to follow the contents of this thesis. We introduce anomaly and intrusion detection terminologies, provide an overview over the most popular datasets used in the field, present common IDS tools and finally, we describe the machine-learning models and techniques we used throughout this thesis.

In Chapter 3, we define the design goals of the thesis and describe the workflow we followed to reach them.

In Chapter 4, we explain how we obtained labels for the C&C-sessions in our datasets and how we developed the set-up we use for analysing this data.

In Chapter 5, we describe the supervised and unsupervised machine-learning models we developed in this thesis, while documenting the model-selection as well as the feature-selection process. We also provide a thorough analysis of the robustness of the final models.

In Chapter 6, we present the results from the extensive evaluation of the developed models.

In Chapter 7, we summarize the main contributions made in this thesis and provide an outlook containing possible future research directions.

Chapter 2

Background

In this chapter, we introduce the background knowledge that is required to follow the approach we pursue in this thesis.

In Section 2.1 we describe the anomaly and intrusion detection problem and introduce commonly used terminologies. We further provide a list the most popular datasets being used in the field while commenting some of their shortcomings. In Section 2.2 we provide an introduction to packet-level and flow-level based analysis. Section 2.3 contains a description of the tools Bro [67] and Snort [76] which we used for network data analysis. We provide information about the Locked Shields cyber-defense exercise in Section 2.4, and in Section 2.4.2 we describe the Cobalt Strike framework, being one of the main penetration tools used by the red team during the exercise. Finally, in Section 2.5 we describe the machine-learning models and techniques we used in this thesis.

2.1 Anomaly and intrusion detection in networks

2.1.1 Overview

Intrusion detection methods target the problem of finding exceptional patterns in network traffic that do not conform to the expected normal behavior. An event or an object is defined as anomalous if its degree of deviation with respect to the normal profile or behavior of the system is high enough. Anomalies appear due to network intrusions and attacks, but also because of malfunctioning devices or overloads in the network [39]. As the topic of this thesis focuses on the intrusion detection domain, in the following the term anomaly will refer to exceptional patterns in the data caused by network intrusions.

In the literature, often the distinction between **misuse-based** and **anomaly-based** intrusion detection is made. Misuse-based systems search for known intrusive patterns in the data based on known signatures while anomaly-based intrusion detection aims to identify unusual patterns the data without relying completely on prior knowledge and signatures.

Anomalies can be classified into the following three categories:

1. Point Anomalies "If an individual data instance can be considered as anomalous with respect to the rest of data, then the instance is termed as a point anomaly. This is the simplest type of anomaly and is the focus of majority of research on anomaly detection. [16]" In Figure 2.1 we observe two big clusters of datapoints N_1 and N_2 while the point anomalies o_1, o_2 and O_3 lie outside the boundaries of the most-frequent/normal regions.

We can further distinguish between **local** and **global** point anomalies. While the global anomaly o_1 clearly lies far away from the normal clusters, the local anomaly o_2 on the other hand is located fairly close to the N_2 cluster. However the distance from o_2 to the points in N_2 is significantly bigger than the mean distances between the points belonging to the cluster, which is why we can identify it as anomaly.

2. Contextual Anomalies A contextual anomaly can only be identified as such, when including information about the context where the anomaly occurs into the analysis. In time-series contextual anomalies can be identified by comparing points and repeating patterns at different point in time.

3. Collective Anomalies The individual data instances in a collective anomaly may not be recognized as anomalous by themselves, but only by taking the occurence of related instances into account [16]. This anomaly-type also commonly applies to time-series, but in contrast to contextual anomalies, a collection of points has to be compared to the rest of the data in order to detect them as such.

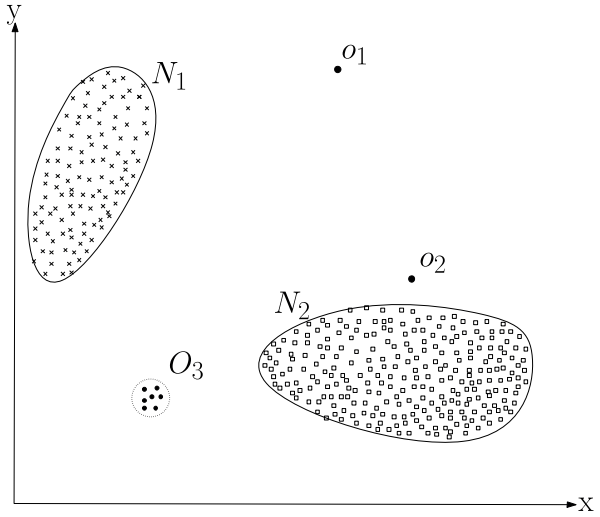


Figure 2.1: An example of anomalies in a 2-dimensional data set [16]. While the *global* anomaly o_1 lies far away from the normal clusters N_1 and N_2, the o_2 on the other hand is located close to the N_2 cluster which defines it as a *local* anomaly.

Anomaly and intrusion detection systems as used in cyber-security incidents can conventionally be categorized into two different types, namely host-based and network-based intrusion detection and prevention systems.

Host-based Intrusion Detection System (HIDS) HIDS aim at collecting information about activity on a particular single system, or host [9]. These systems work by collecting data about events taking place on the system where the IDS is deployed. A main disadvantage is that these systems can be compromised once the host gets infected after a successful attack.

Network-based Intrusion Detection System (NIDS) NIDS collect information from the network itself by capturing network traffic on the wire rather than from each separate host. These systems check for attacks or suspicious behaviour by inspecting the contents of the packets or calculating statistics of the flows moving across the network. A major challenge is the scalability of NIDS in large scale and high-speed networks, where huge amounts of data can arise. This often requires powerful computational resources, depending on the complexity of the NIDS being used as well as on the throughput on the intercepted wires.

A combination of these two techniques provides the best possible protection of a network. However, as in this thesis we deal with data originating from network traffic recordings, in the following we will focus on network-based rather than host-based approaches.

2.1.2 Available datasets

In order to compare different investigations, the availability of public benchmark datasets and the evaluation of the proposed methods on this data is paramount. This section gives a brief overview over the IDS datasets that have been used most in past researches. To evaluate and

interpret the results of past researches in this domain, it is crucial to know the data used in these studies and to be aware of their peculiarities and shortcomings. However, note that due to the variety of networks, traffic profiles and attacks, the representativeness of any network traffic data set can be questioned [39].

Even though our experiments rely mostly on new and not publicly available datasets originating from two past Locked Shields cyber-defense exercises, the selection of the techniques we use is strongly inspired on past research, particularly by focusing on methods that have shown to perform well on the available benchmark datasets.

KDDcup99 In 1998 researchers at Lincoln Laboratory constructed the so called *DARPA* dataset designed for network security analysis applications. Besides benign traffic such as email, browsing, FTP, Telnet, IRC, and SNMP activities, it contains attacks such as DoS, Guess password, Buffer overflow, remote FTP, Syn flood, Nmap, and Rootkit which were all artificially injected during accumulation of the data. While *DARPA* consists of raw *tcpdump* packet captures, later, in 1999 researchers from the University of California generated the *KDDcup99* dataset [42] by processing the *DARPA* tcpdump portion and extracting 41 different network traffic features.
The *KDD* dataset consists of approximately five million single connection vectors, each of which is labelled as either normal or attack with a specific attack type. Despite severe critics being raised in 2000 by McHugh et al. [54], the *KDD* dataset has been the most widely used dataset for the evaluation of network anomaly detection methods and systems. The redundant records in the train set, a huge amount of duplicate records in the train and test sets as well as the synthetic generation of the background traffic in the *DARPA* data have been accounted to be some of the main shortcomings of this benchmark dataset.
Today, the *KDDcup99* dataset is considered to be outdated and should not be used anymore for the effective evaluation of IDSs on modern networks [84].

NSL-KDD In 2009 Tavallaee et al. presented a new version of the *KDD* dataset *NSL-KDD* [84], attempting to solve some of the major issues that led to very poor evaluation of anomaly detection approaches. Tavallaee et al. noted that their dataset still may not be a perfect representative of existing real networks. However they believed that because of the lack of public data sets for network-based IDSs, it still could be applied as an effective benchmark data set to help researchers compare different intrusion detection methods [84]. Until today *NSL-KDD* remains one of the most prominent benchmark dataset used in the network anomaly and intrusion detection domain.

DEFCON Started in 1992 by the Dark Tangent, DEFCON [22] is the world's longest running and largest underground hacking conference. Over the years several IDS datasets have been acquired within the conference. One of the most prominent of these data sources originates from the conference in the year 2000 and is commonly refered to as *DEFCON 2000* or *DEFCON-8* dataset [22]. It contains port scanning and buffer overflow attacks. Another dataset - *DEFCON-10* - commonly used for evaluation of IDSs was created in 2002 during a Capture The Flag (CTF) competition. The dataset contains port scans and sweeps, bad packets, administrative privilege, and FTP by telnet protocol attacks [35]. However, the traffic produced during CTF is very different from real world network traffic, which makes its suitability for real intrusion detection tasks questionable [12].

CAIDA The Center of Applied Internet Data Analysis has collected several networking and IDS datasets [32], most of them being very specific to particular events or attacks. However, due to a number of shortcomings, several investigations have denied the suitability of *CAIDA* as effective IDS dataset [35].

CDX The *CDX-2009* dataset [2] originates from the annual CDX network warfare competition led by the NSA. "In this dataset common attack tools namely Nikto, Nessus, and WebScarab have been used by attackers to carry out reconnaissance and attacks automatically. Benign traffic includes web, email, DNS lookups, and other required services. CDX can be used to test IDS alert rules but it suffers from the lack of traffic diversity and volume [35]."

Kyoto The *Kyoto* dataset [80] was published in 2009 and contains traffic captures from Kyoto University's honeypots. The benchmark data consists of 24 statistical features; 14 conventional

features based on *KDDcup99* and 10 additional features. The background traffic was simulated during the attacks, however, containing only DNS and email protocols.

ISCX2012 In this dataset [74] acquired by the University of New Brunswick, real traffic traces are analysed to create profiles for agents that generate realistic traffic for HTTP, SMTP, SSH, IMAP, POP3, and FTP protocols. Various multi-stage attacks scenarios were subsequently carried out to supply the anomalous portion of the dataset. However, the data set fails to represent new network traffic mainly due to the lack of HTTPS which accounts for nearly 70% of today's network traffic [73].

CICIDS2017 After analysing the common mistakes and criticisms of other synthetically created datasets, Gharib et al. from the University of New Brunswick identified eleven criteria that are necessary for building a reliable benchmark dataset [35]. In 2017 they presented the *CICIDS2017* dataset, which they claim to be the first IDS dataset fullfilling all of the eleven criteria. The dataset contains benign traffic based on HTTP, HTTPS, FTP, SSH, and email protocols and some of the most up-to-date common attacks such as Brute Force FTP, Brute Force SSH, DoS, Heartbleed, Web Attack, Infiltration, Botnet and DDoS [73].

2.2 Packet-level vs. flow-level based traffic analysis

Network traffic analysis aims to deduce meaningful information from patterns in the communication observed on the network wires [79]. There are two main approaches to perform traffic analysis on network data; packet-level analysis and flow-level analysis.

Packet-level analysis Packet capture involves the interception of network packets as they move through a network. Managed switches that support port mirroring can mirror the network packets seen on one switch port or a specific VLAN to a network monitoring connection on another port [48]. Capturing tools such as *tcpdump* [34] can then be used to store the copy of the traffic to a storage device.

After capturing the packets, a Deep Packet Inspection (DPI) solution such as *Wireshark* or *NetworkMiner* can be used to browse the packet contents and to facilitate the analysis of the data.

Flow-level analysis In this work we define a **flow** or **connection** as a series of packets exchanged between two hosts, identifiable by the 5-tuple (source address, source port, destination address, destination port, transport protocol). Note that this definition can vary depending on the flow standard being used (e.g. *NetFlow* [18], *IPFIX* [20]). Many routers and switches operating at OSI-Layer 3 provide flow export options enabling flow aggregation in real-time [39].

Another option is to extract flows in an offline fashion by processing PCAP [85] traffic captures. The stored packets are parsed and aggregated into flows according to the defined flow standard.

Flows can be aggregated either in a bidirectional or unidirectional fashion. **Bidirectional flows** define a forward (source to destination) and backward (destination to source) direction, where the source and destination IP addresses (also called originator and responder addresses) of a bidirectional flow are defined as the source and destination observed in the IP-layer of the first packet in the flow. The packets observed both in the forward and the backward direction will be aggregated into the same flow. In **unidirectional flows** the packets observed in both directions will result in two separate flows as every flow record contains the attribute of a single endpoint only.

Flow analysis can be advantageous for performing high-level analysis while packet inspection can provide more detail which might be relevant for certain applications, for instance the in depth analysis of a particular connection. However, full packet capturing and analysis can be somewhat expensive on busy networks and in case of encrypted traffic the packet payloads will often not provide much meaningful information to the analysis.

2.3 Network analysis and IDS tools

In this Section we provide a description of the common IDS tools Bro [67] and Snort [76] which we both used during the analysis of our datasets.

2.3.1 Bro

Bro [67] is one of the most popular network analysis framework and IDS tools used in the domain, supporting both real-time and offline flow extraction and analysis. It follows the 5-tuple based definition for connection such as outlined in Section 2.2 to accumulate the raw packets into flows. Bro gathers information about flow statistics, such as duration, transferred packets/bytes and connection state. The tool includes a sophisticated signature-based protocol analysing framework which can recognize a variety of different protocols and services without relying on port numbers solely. While UDP and ICMP connections are simply interpreted as a sequence of packets matching the 5-tuple based flow semantics observed within a specified timeout window, Bro differentiates between various different states for TCP flows. Table 2.1 provides a comprehensive list of the TCP state definitions. To extend and customize the functionality and features of the framework, Bro includes an event-driven scripting language that can be used to implement custom scripts and analysis tools.

S0	Connection attempt seen, no reply.
S1	Connection established, not terminated.
SF	Normal establishment and termination. Note that this is the same symbol as for state S1. You can tell the two apart because for S1 there will not be any byte counts in the summary, while for SF there will be.
REJ	Connection attempt rejected.
S2	Connection established and close attempt by originator seen (but no reply from responder).
S3	Connection established and close attempt by responder seen (but no reply from originator).
RSTO	Connection established, originator aborted (sent a RST).
RSTR	Responder sent a RST.
RSTOS0	Originator sent a SYN followed by a RST, we never saw a SYN-ACK from the responder.
RSTRH	Responder sent a SYN ACK followed by a RST, we never saw a SYN from the (purported) originator.
SH	Originator sent a SYN followed by a FIN, we never saw a SYN ACK from the responder (hence the connection was “half” open).
SHR	Responder sent a SYN ACK followed by a FIN, we never saw a SYN from the originator.
OTH	No SYN seen, just midstream traffic (a “partial connection” that was not later closed).

Table 2.1: TCP states in *Bro* [66].

2.3.2 Snort

Snort [76] is an open source network-based intrusion detection framework with the ability to perform real-time as well as offline traffic analysis. It uses predefined rules in order to detect malicious activity on the network. While *Bro* focuses more on building up extensive knowledge about the network traffic, Snort is mostly used as an alert system to automatically inform the system administrator and to block malicious traffic matching the patterns defined in the deployed rule set. More information about Snort and rule-based IDS in general is provided in Section 1.2.2.

2.4 Locked Shields

2.4.1 The exercise

Locked Shields is one of the largest and most complex international cyber defence exercise in the world, organized by the NATO-affiliated Cooperative Cyber Defence Centre of Excellence (CCDCOE), which has been organising the event since 2010 in Tallinn, Estonia [60].

In Locked Shields 2017, the security response teams of the participating nations have been tasked to maintain the services and networks of a military air base of a fictional country targeted by massive and coordinated cyber-attacks launched by the hacker-team of the exercise. The attacks aim for critical infrastructure, including Internet services provider, power grids, military airbase and telecommunication networks.
Locked Shields focuses on realistic and cutting-edge technologies, networks and attack methods [60].

In the following we describe the different teams involved during the exercise:

Red Team The Red Team's mission is to compromise or degrade the performance of the systems in the network, pursuing a number of different objectives. Red Team members are not competing with each other, their objective is to conduct equally balanced attacks on all the Blue Teams' networks. The Red Team uses a white-box approach, meaning that the technical details of the initial configuration of the Blue Teams' systems are available to the Red Team beforehand [21].

Blue Team The Blue Teams represent the security response teams of the participating countries, whose main task is to secure and protect the network infrastructure against the Red Team's attacks. Unlike the Red Team, the information about the initial configuration of the network was not provided to the Blue Team before the exercise. However, during a limited time period before the exercise, the Blue Teams are given the opportunity to connect to the network in order to investigate the given architecture. While the organisers of the exercise are operating on-site in Tallinn, Estonia, the participating Blue Teams work remotely through secure connections from their home bases.

Yellow Team The Yellow Team's role is to provide situational awareness about the game to all other participants. The main sources of data for the Yellow Team were reports provided by the Blue Teams, reports on the status of attack campaigns received from Red Team members, and the results of automatic and manual scoring during the exercise [21].

In addition, the **White Team** is responsible for preparing the exercise and controlling it during Execution whereas the **Green Team** is in charge of setting-up the the technical infrastructure and maintaining functionality during the exercise.

2.4.2 Cobalt Strike

Cobalt Strike (CS) [49] is an adversary simulation and penetration testing framework, designed to issue targeted attacks and to execute post-exploitation actions of advanced threat actors.
CS is one of the main penetration tools being used by the red team during the Locked Shields exercise. It provides a number of different exploit techniques including user-driven attacks, web drive-by attacks and it can be combined with the popular *Metasploit* [68] framework in order to exploit other known vulnerabilities.

2.4.2.1 Command and Control (C&C)

After running a successful exploit, CS can be used to used to execute commands on the compromised host. CS's payload for long-term asynchronous command and control of compromised hosts is called **Beacon**. Beacon downloads tasks issued by the threat actor to the compromised machine using HTTP(S) or DNS requests.

When generating a new Beacon payload in CS, the threat actor can specify the IP-address or domain-name of the C&C server that delivers the tasks to the compromised host and redirects the output of the issued commands back to the attacker.
Once a C&C session is open, the compromised host will contact the specified C&C server in a periodic fashion to check for new directives.

In the Locked Shields 2018 exercise, five shared and one private C&C servers per member were assigned to the read team during the exercise. Each attacker would generally use their own private team server to generate and host Beacon payloads for execution on the target hosts, and then replicate any high-privilege access that was gained to the shared team servers [1].

In the following we describe how a typical Beacon HTTP Transaction looks like:

1. Each transaction starts when the compromised host sends a HTTP GET request to the C&C server. The request contains metadata with information about the compromised system.
2. The C&C server responds to this HTTP GET request with commands that the compromised host must execute. The packet payload containing these commands is encrypted.
3. As the task are executed, Beacon accumulates the corresponding output. After executing all tasks, the compromised host will deliver the output to the C&C server by sending a HTTP POST message with encrypted payload.

Note that the HTTPS and DNS communication schemes function in a similar manner [49].

In this thesis we usually refer to Cobalt Strike's Beacon sessions as C&C sessions because the terminology is commonly understood also by people not familiar with the Cobalt Strike framework.

2.5 Machine-learning

In this section, we provide background knowledge about the machine-learning principles and algorithms we use throughout this thesis.

2.5.1 Supervised vs. unsupervised learning

Within the field of machine learning, we can distinguish two main types of different tasks; supervised and unsupervised learning.

The training process of supervised models relies on the availability of a ground truth ("labels"), about the data samples. The goal of supervised learning is to learn a function that approximates the relationship between the samples and the corresponding labels in the data. A common example is the classification of cat and dog images. During the training phase of the model, we feed many pictures into the model while telling the model if a picture shows a cat or a dog. After finishing training, the model should be able to classify new unseen images correctly.

The unsupervised approach on the other hand does not rely on labels. Clustering is one of the most popular problems in the unsupervised domain. The goal is to to infer the natural structure present in the data and to detect groups ("clusters") of samples that are similar to each other. In the above example, an unsupervised model would create two clusters, one containing dog images and the other cats, however, the model is not told about what these images show during the training process.

2.5.2 Defining the training, test and validation sets

In machine-learning it is commonly recommended to split the data into three different subsets; training, test and validation sets.
The training set is used to teach the models to perform the desired task. After each training iteration the performance of the model is evaluated on the validation set, which is used for tuning

the model's hyper-parameters. After finishing the tuning process, it is strongly recommended to perform a final evaluation on the test dataset, consisting of a set of previously unseen samples neither during training nor during the evaluation within the parameter tuning process.
This is crucial as in the hyper-parameter optimization loop we might overfit the parameters of the model to the validation set, as we used the prediction quality on this set as criterion for the quality of the chosen parameters.

Another common method to counteract the overfitting problem is to apply **K-fold cross validation**. For instance using a 3-fold we would split our data into 3 equally sized subsets A,B,C. We then have three options to choose the train and validation sets:

1. Training on [A,B] validation on [C]
2. Training on [B,C] validation on [A]
3. Training on [A,C] validation on [B]

Repeated K-fold cross-validation describes the same process but repeated several times while shuffling the dataset after each iteration, leading to new folds in each repetition.

2.5.3 Pre-preprocessing

2.5.4 Standardization

Many machine-learning algorithms make assumptions about the dataset used for training. One common issue in datasets is that the value sizes and variances of the different feature vectors might differ by multiple orders of magnitude, depending on the measuring unit the features describe. For instance one feature might encode a time measure in seconds while another feature describes a count value. Depending on the algorithm being used, a preprocessing step to bring the different features to the same scale can be crucial.

Standardization is a method widely used for rescaling in many machine-learning tasks. The features are scaled in such a way that they have a mean value of 0 and a standard deviation of 1:

$$x' = \frac{x - \overline{x}}{\sigma} \tag{2.1}$$

$x =$ Feature vector
$\overline{x} =$ Mean value of x
$\sigma =$ Standard deviation of x

2.5.5 Dealing with categorical features

Another common issue in machine learning is the presence of categorical/nominal features, as many algorithms require all input variables and output variables to be numeric. So to make use of categorical data, it has to be first converted into a numerical form. Simple enumeration of the categorical values is often not sufficient. Lets assume the feature 'protocol' that can take the values TCP, UDP and ICMP and we assign them the values 0, 1 and 2 accordingly. Doing so, in a numerical sense ICMP is now further away from TCP than from UDP, as the euclidean difference is bigger. However, this differentiation is introduced by the enumeration and was not inherent in the original categorical definition of the features.

A common solution to this problem is to apply **one-hot encoding** to the categorical features transforming every nominal value of a particular categorical feature into a new binary feature that can equal 0 or 1. In the above example we would transform the protocol feature to three new binary features TCP, UDP and ICMP. If the TCP feature has value 1, this is equivalent to the original protocol feature being TCP. The main drawback of such a mapping is that it can cause a dramatic increase in dimensionality if a categorical feature takes on a big number of different nominal values. Also the data becomes more sparse, training and inference runtimes

increase due to the higher dimensionality and the original balance among the features might get distorted.
On the other hand, approaches based on decision trees such as random forest [14] can work with categorical data directly.

2.5.6 Supervised algorithms

In the following we briefly present the supervised classification algorithms that we used throughout this thesis. The aim is to give an overview about the function of the different algorithms while pointing out their advantages as well as possible limitations.

2.5.6.1 Random Forest

Random forest classifiers represent ensembles of multiple decision tree predictors. The prediction is inferred by the majority vote of the trees contained in the ensemble.
In order to explain random forest, we first have to describe how decision trees work in general. Decision tree algorithms solve the classification problem by using a tree representation. Each node of the tree represents a learned decision rule that depends on a particular feature of the data instance that is to be classified. The leaf nodes on the other hand correspond to the labels of the dataset. So depending on the feature values of the data instance to be classified, it will take a different path from the root to one of the leaves in the tree. The final prediction is inferred from the class belonging to the leave where the sample ends up after traversing the tree.

However, classifiers based on single decision trees have high potential of overfitting the training data. Furthermore, robustness is an issue as variations in the data might result in a completely different tree.
Random forest was developed on the observation that significant improvements in classification performance can result by growing an ensemble of trees and letting them vote instead of relying on the predictions of one single tree only [14].
The diversity among the trees in the ensemble is created by introducing randomness during the classifier construction, where different trees are grown on various randomly subsampled sets of the original data.

The main advantages of random forest models are listed in the following:

1. Little parameter tuning required for good performance
2. Classifier performance is invariant to strongly correlating features
3. Highly scalable due to linear computational complexity for training and logarithmic complexity for inference.
4. Can deal with unbalanced datasets
5. Can deal with categorical features

2.5.6.2 Support Vector Machine (SVM)

SVM is a supervised classification algorithm, which aims to find the hyperplane that separates the classes in the data while maximizing the distance margins to the class instances.
Figure 2.2 shows an example of the separating hyperplane and the separation margin of a SVM.

A major limitation of linear SVMs is that they perform well only on data where the instances are linearly separable, such as in Figure 2.2 where we can draw a straight line to partition the two classes. By applying the so called kernel trick, where a non-linear mapping function is applied to the features, SVMs can perform also on non-linear separable datasets. However, this increases the complexity to $O(n^2)$ or even $O(n^3)$ depending on the kernel function being used.

In this work we only investigate the linear SVM approach, as we found that the kernelized version does not scale well on our data.

Further limitations of the algorithm are its sensitivity to correlating features and its inability to deal with categorical features.

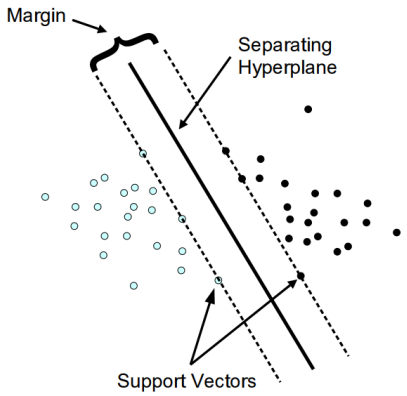


Figure 2.2: Illustration of the separating hyperplane and the separation margin of a SVM [56].

2.5.6.3 Logistic Regression

Logistic Regression [13] is a classification algorithm that aims to find linear decision boundaries between the different classes in the data. In this aspect it is very similar to the SVM model as described above. However, while SVM maximizes the margin between the points closest to the boundary, Logistic regression models have a probabilistic interpretation and are trained to maximize the conditional probability between the labels and samples of the training data. So the output of the model is a probability value expressing the likelihood for a particular sample to belong to a certain class.

The linear complexity makes the algorithm scalable, and the probabilistic output facilitates the integration into probabilistic frameworks. The prediction quality generally is comparable to the performance of linear SVMs.

2.5.6.4 Naive Bayes

Naive Bayes [13] is a probabilistic classifier that applies Bayes Theorem, assuming that the features are conditionally independent to each other.
Despite the fact that the independence assumption rarely holds, Naive Bayes models can still achieve good performance even in cases where the assumption is violated.

A typical assumption is that the continuous values associated with each class are distributed according to a Gaussian distribution. However, due to this assumption, the algorithm fails to perform well on datasets where the distribution of the classes is not gaussian, which is also the main limitation of this approach.

The main advantages are that Naive Bayes classifiers are fast and highly scalable.

2.5.6.5 K-Nearest Neighbours (K-NN)

K-NN is a very popular supervised algorithm used in classification tasks.
As the name suggests, the algorithm bases its prediction on calculating the K nearest neighbour points to the sample that is to be classified. The class that occurs most frequently in these K neighbour points is used to classify a new instance. However, this means that the whole training data needs to be stored, as the inference process relies directly on the training samples. The prediction process is illustrated in Figure 2.3.

While K-NN performs well in variety of different task it is slow when running predictions as the distances between the new data and all the training data samples need to be computed. The bigger the training set being used, the slower the inference process. This is a major issue in real-time classification tasks where small latency is crucial. Furthermore, the K parameter has to be set before training to a value suitable to the dataset being used. Note that in the example depicted in Figure 2.3, the prediction outcome will be different for K=1 and K=3.

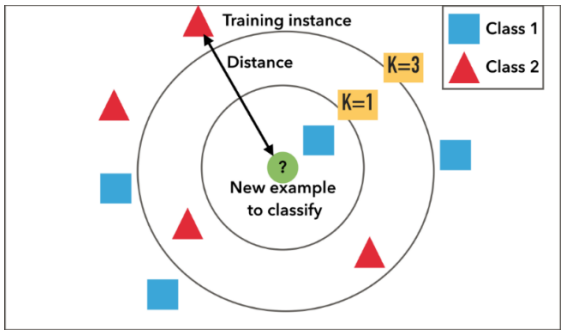


Figure 2.3: Illustration the K-NN algorithm. Illustration from [15].

The computational complexity of the algorithm is quadratic and the memory requirement grows linearly with the size of the training data, so this algorithm does not scale well to big data.

2.5.7 Boxplots

In Chapters 5 and 6 we will use boxplots [52] to report the performance metrics of our classifiers. In the following, we briefly explain how they can be interpreted.

Figure 2.4 shows an example of a boxplot for a generic metric that can lie between 0 and 1, such as accuracy. Let's assume that we run 10 consecutive experiments, while measuring this metric. Visualizing these results in a boxplot is a way to illustrate the distribution of the measured values. The orange horizontal line in Figure 2.4 represents the median over the 10 measurements. The median is generally preferred over the mean value, because it is less sensitive to outliers. Q1 and Q3 represent the first and third quartiles of the measurement. The first quartile is the median of the lower half of the data set. This means that about 25% of the numbers in the data set lie below Q1 and about 75% lie above Q1. The third quartile is the median of the upper half of the data set. This means that about 75% of the numbers in the data set lie below Q3 and about 25% lie above Q3.

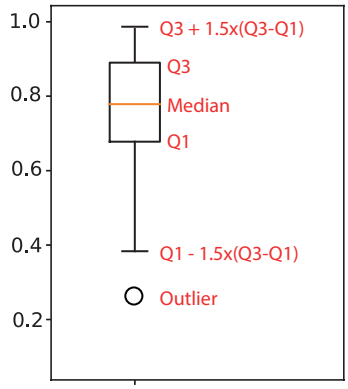


Figure 2.4: Example of a boxplot for a generic performance metric.

2.5.8 Unsupervised clustering algorithms

In this section we describe the unsupervised clustering algorithms that we used in this thesis.

2.5.8.1 K-means

K-means [40] is one of the most commonly used unsupervised clustering algorithms. It is a method to automatically partition a data set into K different groups.

Before running the algorithm, the desired number of clusters K has to be configured, and in the initialization phase, K cluster-centers are distributed in the data space either randomly or by following more sophisticated seeding strategies such as K-means++ [6].
After defining the initial cluster-centers, the algorithm will repeat the following two steps until convergence:

1. Each data point gets assigned to its closest cluster center by measuring euclidian distance.
2. Each cluster-center is updated by calculating the mean of all data instances that have been assigned to it in step 1.

The algorithm converges when the assignments of the data points to the cluster-centers in step 1 do not change anymore.

The main limitations of the algorithm are the following:

1. The number of clusters K needs to be defined first.
2. It can only find clusters with spherical shape.
3. The outcome depends strongly on the initialization of the cluster-centers. Convergence to a global optimum is not guaranteed.
4. It can not deal with categorical features as it relies on a euclidean distance metric.

Figure 2.5 shows two popular examples of 2-dimensional datasets where K-means fails due to the limitations listed above.

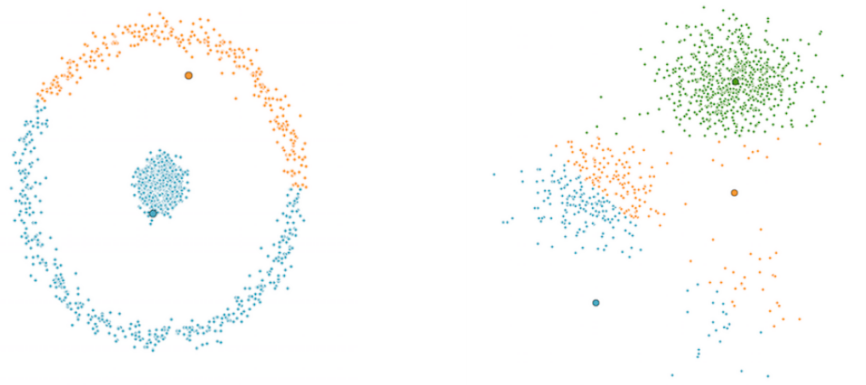


Figure 2.5: Two examples of 2-dimensional datasets where K-means fails. Pictures taken from [51].

Despite these drawbacks, K-means is widely used and is one of the very few clustering algorithms with only linear time complexity, which enables applications in big data without scaling issues.

2.5.8.2 DBSCAN

DBSCAN [71] is a density based clustering algorithm. It groups data points in high-density areas, meaning that the samples in these areas occur with many close neighbours. Points that do not occur in such areas are classified as outliers.

Before training, two important parameters have to be configured; Epsilon and minSamples.
Epsilon is a radius that specifies the minimum distance between two points such as to be considered a part of a cluster. MinSamples measures how many neighbours a point needs to have to be included into a cluster.

In DBSCAN, the density associated with a point is obtained by counting the number of points within the specified Epsilon radius. Points with a density value above the threshold MinSamples are classified as core points [29]. Points that do not have any core point within the specified radius will be classified as outliers.

The main strength of DBSCAN is that it can identify clusters of arbitrary shapes and sizes. The main limitation on the other hand is that it cannot handle clusters of differing densities due to its density based definition of core points [29]. Also, DBSCAN has quadratic computational complexity and thus does not scale well to big datasets.

Chapter 3

Overview

In this section, we provide a high-level overview of the design goals and our workflow describing the chosen approaches to achieve these goals.

3.1 Design goals

In this investigation we aim to achieve the following goals.

Goal 1: Large scale network data analysis Find ways to effectively analyse and visualize network traffic data on a large scale and on different levels of abstractions.

Goal 2: Identify malicious activity patterns Investigate and develop statistical and machine-learning based methods to detect malicious activity in network traffic data.

Goal 3: Develop an intrusion detection system Throughout this thesis, we use network traffic captures from the international Locked Shields 2017 and 2018 cyber-defense exercises. We aim to develop a system that supports the analysts in detecting malicious activity during future competitions.

3.2 Workflow

Figure 3.1 illustrates a high-level abstraction of the workflow that we followed to meet the design goals specified above. In the following we provide a brief description for each of the building blocks contained in this diagram.

Data Enrichment In the Data Enrichment phase we generate different representations of the Locked Shields data. As outlined in Section 2.2, having a flow-based representation of the network data rather than looking at the raw PCAP data directly, enables the analysis of network traffic data on a higher level of abstraction. This is crucial when performing data analysis on a large scale. To extract the flows from the PCAP data, we use the common flow-extraction and IDS framework Bro [67] (see Section 2.3.1).

In addition, we run the PCAPs through the popular IDS tool Snort [76] to generate alerts for possible malicious candidates.

Following this procedure we obtain three different representations of Locked Shields data; Packets, Flows, Alerts.

Feature Extraction In the Feature Extraction stage, we generate two different sets of flow-based network traffic features, that we will later use to train our final machine-learning models.

First, we extract the features that are used in the most popular intrusion detection dataset for machine-learning; KDDcup99 (see Section 5.1.1).

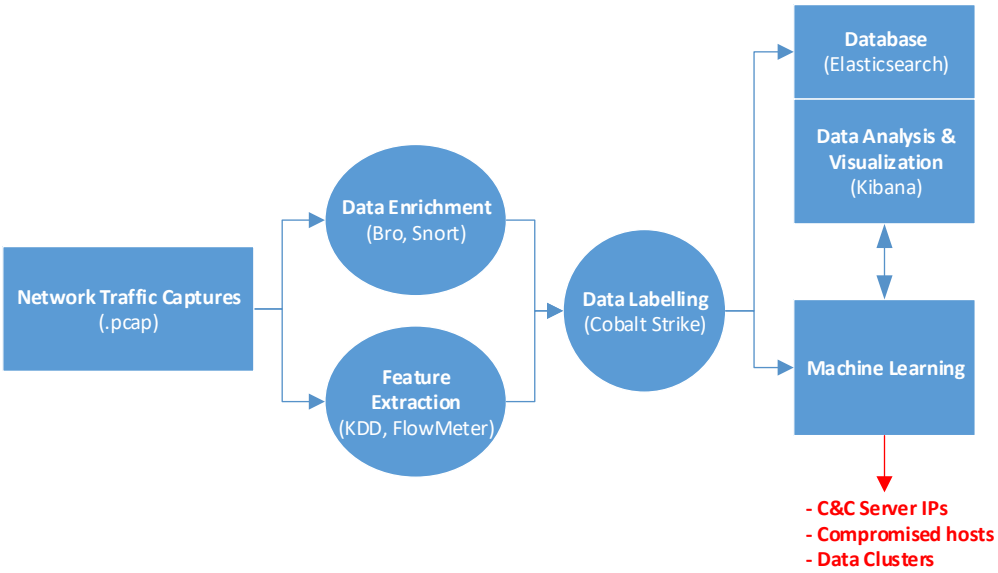


Figure 3.1: High-level illustration of the workflow followed in this thesis

Second, we use a modified version of the FlowMeter feature-extraction tool (see Section 5.1.2) to generate the traffic features recently proposed by [73].

In the machine-learning part of the thesis, we evaluate the performance of the obtained models comparing these two sets of features.

Data Labelling In the next stage we label the data using reports automatically generated by the Cobalt Strike penetration testing framework which recorded the activity of the the red team during the Locked Shields exercise. In particular, we label flows corresponding to C&C sessions between compromised machines and the C&C servers owned by the red team.

Database We build up an extensive database using *Elasticsearch* [23], merging all the different levels of abstraction we created in the Data Enrichment stage. We then use Kibana on top of the Elasticsearch database as a tool to search, view, and interact with the data as well as to create meaningful visualizations of the recorded traffic.

Data Analysis & Visualization Using the Elasticsearch set-up we developed, we conduct a thorough analysis of the Locked Shields data. In Section 4.4.1.5 we provide two showcases where identify DoS and SQL injection attacks using this architecture.

Machine Learning In the Machine Learning part of the thesis we first train supervised classifiers using the labelling of the C&C sessions we obtained in the Data Labelling stage. We show that the trained models predict flows corresponding to such sessions with high precision. We then analyse the robustness of the obtained models by generating adversarial inputs and simulating packet loss.

Finally, we evaluate unsupervised clustering algorithms both on the novel intrusion detection dataset CICIDS2017 [73] as well as on the Locked Shield datasets. We show the feasibility of detecting malicious activity in both datasets in a fully unsupervised fashion.

Chapter 4

Network data analysis

In this thesis, we work with two large sets of raw network data originating from the Locked Shields exercises in 2017 (LS17) and 2018 (LS18). The raw PCAP-files of the corresponding two years' challenges are approximately 114GB and 216GB in size. This chapter describes the methodology we follow and the design choices we have made to analyse this data. In Section 4.1 we describe how we apply the common IDS tools Bro and Snort to obtain flow and alert-based representations of our data. We then explain in Section 4.3 how we used the automatically generated reports by the Cobalt Strike framework to label the C&C sessions in the Locked Shields traffic captures. Finally, in Section 4.4 we describe the development of our network data analysis set-up by building up an extensive Elasticsearch database, containing the different representations (packets, flows, alerts) of the network data.

4.1 Data enrichment

In this Section we document how we use the common IDS tools Bro [67] and Snort [76] to obtain flow and alert-based representations of the LS17 and LS18 PCAP data.

Bro To obtain a flow-based representation of our raw PCAP data, we used the network analysis tool Bro, which we described in Section 2.3.1. Bro generates a variety of different log files stored in comma-separated (CSV) format. The *conn.log* file is of particular interest to us as it summarizes all TCP, UDP and ICMP connections and provides thus a flow-based view on our data. In addition, Bro's protocol analysers (e.g. dhcp, dns, ftp, http, ssl) provide more detailed information on the activities of the observed protocols and services. We used a custom script *sslAnalyzer.bro* [55], which generates a comprehensive list of all SSL events observed on the wire, containing the corresponding IP-addresses, ports and host-names.

Snort As we do not have complete information about the malicious activities performed by the red team during the exercise, we decided to use Snort, one of the most deployed IDS tools in industry, to obtain a first view on the abnormal traffic in the data. We deployed Snort using the *Community* and the *Registered* rule-sets, which are both freely available [77].
One of the main difficulties in the deployment of rule-based IDS systems such as Snort, is that in order to make them reliable and to keep false alert rates low, fine-tuning of the rules according to the network architecture and the legitimate traffic profiles is required. However, as we perform our analysis in an offline set-up where we analyse recorded PCAP data, the higher false alert rates induced by the lack of rule fine tuning is not as critical as it would be in a online deployment, with the aim of real-time network intrusion detection and prevention.

4.2 Packet duplicates

During the analysis of the Locked Shields data we found that a significant number of the recorded packets is duplicated. In particular, we observe that many packets going from a ma-

chine in the local network (internal) to another machine inside the network, are followed by an exact duplicate, where only the TCP/UDP checksum differs. There is always one packet with valid checksum and a duplicate with invalid checksum. Interestingly we found this peculiarity only in flows going from one internal to another internal host while duplicates in flows going to external servers are absent. We know that during the exercise the network was divided into different Virtual Local Area Networks (VLANs) whereupon the tool *tcpdump* was used to capture the traffic flowing out and into these VLANs. So if the *tcpdump* instance is configured to record the traffic on VLAN1 and VLAN2, and we send a packet from one VLAN to the other, the packet will be duplicated in this configuration. When sending packets from VLAN1 to an external server on the other hand, these will be captured only once if they are not routed through a second VLAN.

It is crucial to take this characteristic of the data into account when extracting the flows and later generating the features to be used to train our final models.
In Bro, the documentation suggests that packets with invalid checksums are discarded by default unless the -C command-line option or toggling the 'ignore_checksums' variable are set. Even so, when running Bro without the -C option, we observed two entries in the generated conn.log file for all TCP flows containing the packet duplicates as described above (see Table 4.1). The first entry is created by the duplicate SYN packet with invalid checksum, leading to a connection entry with OTH connection state and C value in the history field, which indicates the invalid checksum. Note that in this case the definition of the OTH state ('No SYN seen', see Table 2.1) is not accurate, as in fact a SYN packet is seen, but with an invalid checksum. Table 4.2 shows the extracted flow statistics for the same packets but with the -C option set. We observe that the redundant entry for the connection entry disappears while the counts of the number of packets sent by the originator and the responder (orig_pkts, resp_pkts) both doubled. This happens because Bro now increments these counters also for the duplicate packets with invalid checksum. As packets with invalid checksums are by default discarded by the receiving network adapter, we found it to be more meaningful to run Bro in its standard configuration without the -C option, as the redundant connection entries can be easily identified and filtered out if needed.

When processing the data with Snort this peculiarity is not as critical, as the Snort by default will not process packets that have been flagged as having bad checksums.

proto	service	duration	orig_pkts	resp_pkts	conn_state	history
tcp	-	-	0	0	OTH	C
tcp	http	0.05913	5	5	SF	SchCADadfF

Table 4.1: Bro conn.log entries for a single flow with duplicates

proto	service	duration	orig_pkts	resp_pkts	conn_state	history
tcp	http	0.05913	10	10	SF	ShADadftF

Table 4.2: Bro conn.log entries for a single flow with duplicates when taking packets with invalid checksums into account.

4.3 Labelling of the data

For the LS17 dataset, we were provided with three different documents (Indicators of Compromise, Activity Report, Operation Notes) generated by the Cobalt Strike reporting engine, containing detailed information about the activities of the red team during the exercise. For the LS18 dataset we were provided only with a document holding information equivalent to the contents in the Activity Report described in the next paragraph.

In this section, we describe how we made use of this information to label the C&C sessions in the LS17 and LS18 datasets. Note that the red team also used other tools besides Cobalt Strike during the exercise, this activity however, is not investigated in this work.

First, we briefly describe the contents of these reports:

Indicators of Compromise This report first lists some samples of HTTP headers exchanged during transactions exchanged within the C&C sessions opened using the Cobalt Strike framework (see Section 2.4.2). The document further summarizes the MD5 hashes of files uploaded through a Cobalt Strike C&C session. It also lists all IP addresses and domain-names affiliated with the C&C servers used by the red team to communicate with the compromised machines.

Activity Report This document contains a timeline of all red-team activity logged by Cobalt Strike. All activities of the red team within the Cobalt Strike framework are merged into this report. Figure 4.1 shows a brief extract from the activity report. The first line listing the "initial beacon" activity indicates that at this point in time a new C&C session to the listed server was opened. The second activity listed holds a "spawn" command that the a member of the red team issued on the compromised machine in order to replicate any high-privilege access that was gained to the shared team servers (see Section 2.4.2).

26/04 07:37	10.7.3.152	LAB	WS1-02	steve.ballmer	1424	initial beacon
26/04 07:37	10.7.3.152	LAB	WS1-02	steve.ballmer	1424	spawn (x86) windows/beacon_http/ reverse_http (39.82.133.147:443)

Figure 4.1: Extract from Cobalt Strike's Activity Report

Operation Notes This report, just as the Activity Report, shows a timeline of the red team activity but organised on a session-by-session basis. It also summarizes the MD5 hashes of the files uploaded within a particular session as well as the taken communication path (C&C server) of the compromised system to reach the red team. Figure 4.2 shows an extract from this report.

10.7.3.2 (DC3)

Zone: LAB
User: SYSTEM *
PID: 7384
Opened: 27/04 07:18

Communication Path

hosts	port	protocol
apexgames.ex	80	HTTP

File Hashes

date	hash	name
27/04 07:18	524582d1d8279f72c6b2296a0ac536cb	tdsvc.exe

Activity

date	activity
27/04 07:18	cd c:\windows\system32
27/04 07:18	upload tdsvc.exe as tdsvc.exe
27/04 07:18	timestomp tdsvc.exe to cmd.exe

Figure 4.2: Extract from Cobalt Strike's Operation Notes Report

Domain-name resolution The provided IP addresses and domain names affiliated with the red team such as reported in these reports are of particular interest to us. Our approach is to label the flows and packets corresponding to these addresses as malicious activity. To make use of the domain-names, we first resolve them to IP addresses. To do so, we create mappings

between all observed IP addresses and domain-names by parsing the contents in the following log-files generated by Bro: http.log, dns.log and sslanalyzer.log (see Section 4.1). When processing the http.log file, we store the observed domain-names from the http-host fields and the corresponding IP addresses. For DNS, we do likewise using the observed domain-names in the A (IPv4) and AAAA (IPv6) resource records. We then gather the IPs with the corresponding hostnames listed in the recorded TLS hello messages. Finally, we merge the IP to domain-name mappings collected from these three sources and store them as a hash table data structure (python dictionary) to enable fast domain lookups.

As some of the domain-names listed in the Cobalt Strike reports such as google.api.com or news.bbc.com.uk seem trustworthy, we double check the resolution of such domains to prevent the leaking of legit IP addresses into our list of malicious addresses.

In particular, we investigate if all these domains were only used for malicious purposes during the exercise. For instance, in the presence of DNS cache poisoning attacks, it is possible that with our resolving method the affected domain-name would be mapped to its legit IP address before the attack on the DNS server, while the domain might point to a server owned by the red team after a successful DNS attack. However, as all the mappings we obtain are unique for both the LS17 and LS18 data, we can safely rule out this scenario.

In the following, we describe the two different approaches we evaluated in order to label the malicious activities in our data.

Labelling by hosts After resolving the domain-names to IP addresses as described above, we merge them with the other IPs listed in the Cobalt Strike reports to create a comprehensive list of IP addresses affiliated with the red team. We then use this list of addresses and label all connections as malicious where either the source or the destination IP matches one of the addresses in this list.

In the following, we will refer to this type of labelling as "labelling by hosts" or "host-labelling".

Labelling by sessions The Operation Notes report described above lists all the activity performed by the red team within the Cobalt Strike framework on a session-by-session basis. We use this information to obtain a labelling that flags specifically the reported C&C sessions that were opened during the exercise.

If the source and destination IPs of a flow match the addresses of the compromised machine and the communication path listed in the report, and if the connection was opened inside the specified time-window, we label the connection as malicious.

Note that the malicious samples labelled by this approach account for a subset of the labelled samples obtained by the host-labelling method described above, as all the IP addresses in the Operation Notes that we used to match on also appear in the list of all malicious IPs that we used to label the data using the host-labelling method.

We observe that in a couple of the reported sessions there are big time gaps between some of the listed activities. However, as the compromised machines contact the corresponding C&C servers in a periodic fashion (see Section 2.4.2), also when the attacker does not issue any new commands, it makes sense to also label the flows observed in these time gaps.

In the following we will refer to this type of labelling as "labelling by sessions" or "session-labelling".

Note that we found the list of MD5 hashes provided in the reports not to be useful for labelling, as in Cobalt Strikes' Beacon sessions the complete files are not downloaded at once to the compromised machine, but get splitted up into smaller chunks of data [49]. Furthermore, the payloads of the transferred packets are always encrypted, even when using the insecure HTTP Beacon session type for communication (see Section 2.4.2). So what we observe on the wire are encrypted chunks of the transferred files. To match the MD5 hashes listed in the reports in order to flag the corresponding flows, we would need to decrypt and then merge the different chunks to obtain the complete file and then compute the MD5 hash.

4.3.1 Analysis of the obtained labellings

Table 4.3 shows the percentages of the flows extracted by Bro that were labelled as malicious using the two proposed labelling methods for both the LS17 and LS18 datasets. As we were provided with the Operation Notes report only for LS17 and not for the LS18 data, we could obtain the session-labelling only for the LS17 dataset. We observe that the percentage of the malicious samples using the session-labelling method is significantly smaller.

	LS17	**LS18**
session-labelling	0.42%	-
host-labelling	12.77%	4.30%

Table 4.3: Percentages of the Bro flows labelled as malicious under the two proposed labelling approaches: session-labelling and host-labelling.

Our initial motivation to introduce these two different labelling approaches was that while the session-labelling would focus specifically on flows corresponding to Cobalt Strikes' C&C sessions, the host-labelling would be much broader, possibly flagging different types of attacks besides the C&C sessions. However, we observe that in case of the host-labelling, the vast majority of malicious samples is labelled due to the destination IP address of the corresponding flows being in our list of malicious IPs. In the LS17 data only 0.07% of the flows were labelled due to the source IP while in the LS18 data this fraction is even lower at 0.006%, meaning that almost all of the labelled flows were initiated by a compromised machine connecting to a server owned by the red team. This leads us to the conclusion that these flows cannot contain active attacks executed by the red team as such flows would need to go in the other direction; from a red team server to a machine being targeted. That observation brings us to the following hypothesis:

Hypothesis 1 (H1): *The obtained host-labelling mainly labels flows used for communication between already compromised machines and servers owned by the red team (C&C sessions).*

In other words, the host-labelling marks flows corresponding to C&C sessions just as the session-labelling. We found that sessions opened within already existing C&C sessions using Cobalt Strike's *spawn* command, will not lead to a new entry for the newly created session in the Operating Notes report. So these C&C sessions are not labelled using the session-labelling method, which is the main reason why the percentage of labelled malicious flows is significantly lower than for the host-labelling.

Besides the intuitions for Hypothesis 1 given in this section, we provide strong evidence in Chapters 5 and 6 for this hypothesis. Namely, in Section 5.3 we show that using the host-labelling we can train precise supervised classifiers while in Section 5.4 we show that the malicious samples labelled by host represent highly homogeneous clusters in both the LS17 and LS18 datasets.

4.3.2 Snort

So far we have labelled flows corresponding to Cobalt Strike's C&C sessions in the Locked Shields data. However, as outlined above, this labelling does not contain the active exploits executed by the red team during the exercise. Obtaining a complete labelling for all malicious activity in large scale network data is a challenging task without comprehensive prior information.

In order to identify some of the attacks we run the popular IDS tool Snort on our datasets. Table 4.4 shows the alerts raised on the LS17 data ordered by occurrence. We found the false alert rates of the prominent alert-types to be significantly high. For instance, when looking at the *TCP Portsweep* alerts, we observed that a dominant fraction of alerts were raised by flows going from internal hosts out to external servers. While it would be natural to observe portsweeps in the opposite direction targeting machines in our local network, it is unlikely that the red team initiated port scans aiming at external servers during the exercise. On the other hand, less common alert types such as the SQL where only 11 alerts are raised are more likely to be true indicators of malicious activity. However, the sample count in this case (11 vs. total 14mio connections) is far too low for training reliable predictors.

We draw the conclusion that the alerts generated by Snort are not accurate enough to extract a solid ground truth about the malicious activities in order to train our final machine-learning based models. For this reason, we discard the strategy to use the Snort alerts for data-labelling and instead focus on the more accurate labellings based on the Cobalt Strike reports as described above. While we do not use the information provided by Snort for training our machine-learning based models, we still include the alerts into our database, as described in the following section.

Alert-type	Count
(spp_sip) Maximum dialogs within a session reached	76,219
Reset outside window	58,931
(http_inspect) NO CONTENT-LENGTH OR TRANSFER-ENCODING IN HTTP RESPONSE	55,804
Consecutive TCP small segments exceeding threshold	23,617
(portscan) TCP Portsweep	15,289
(spp_sip) Content length mismatch	10,882
(portscan) UDP Portsweep	3,951
(portscan) UDP Portscan	3,925
TCP Timestamp is missing	3,378
(portscan) TCP Distributed Portscan	2,076
(http_inspect) UNESCAPED SPACE IN HTTP URI	1,295
(portscan) TCP Portscan	1,238
TCP Timestamp is outside of PAWS window	809
(http_inspect) LONG HEADER	494
Data sent on stream after TCP Reset sent	408
(http_inspect) NON-RFC DEFINED CHAR	224
(spp_ssh) Protocol mismatch	215
(portscan) UDP Distributed Portscan	117
(http_inspect) INVALID CONTENT-LENGTH OR CHUNK SIZE	101
Bad segment, adjusted size <= 0	91
ACK number is greater than prior FIN	90
(portscan) TCP Decoy Portscan	73
(http_inspect) UNKNOWN METHOD	40
Data sent on stream not accepting data	31
(http_inspect) POST W/O CONTENT-LENGTH OR CHUNKS	24
Limit on number of overlapping TCP packets reached	19
(http_inspect) SIMPLE REQUEST	16
(spp_sip) Invite replay attack	11
SQL 1 = 1 - possible sql injection attempt	10
(http_inspect) HTTP RESPONSE HAS UTF CHARSET WHICH FAILED TO NORMALIZE	6
(portscan) UDP Decoy Portscan	3
OS-WINDOWS Microsoft Windows Active Directory kerberos encryption type downgrade attempt	3
(dcerpc2) Connection-oriented DCE/RPC - Invalid major version: 1	2
(dcerpc2) Connection-oriented DCE/RPC - Invalid major version: 211	1
(spp_ssl) Invalid Client HELLO after Server HELLO Detected	1

Table 4.4: Counts of the alerts raised by Snort on the LS17 data

4.4 Building up the database

After complementing the raw PCAP data from the LS17 and LS18 exercises with a flow-based view of the traffic, the labels obtained from the Cobalt Strike reports and the alerts generated by the popular IDS tool Snort, we now investigate methods to effectively view, filter and illustrate this vast amount of data.

Our approach is to build up a comprehensive database combining the different representations of the data (PCAP, flows, alerts) as generated in the data enrichment and labelling phase of this investigation. This facilitates the forensic analysis significantly as with such a system, we can start the analysis at a higher level of abstraction (alerts, flows) and once narrowed down poten-

tial malicious candidates in the traffic, we can pivot to the more detailed packet-level analysis if further investigation is required.

4.4.1 Elasticsearch

In the following, we describe how we build up the described database using *Elasticsearch* [23]. We have selected the Elasticsearch database program to index our data for the following reasons:

1. Elasticsearch is a noSQL/non-relational database, meaning that data can be stored in a schema-less or free-form fashion. This is crucial when indexing the packet-level data (PCAP), as the contents of different packets can strongly vary among different protocols.
2. Elasticsearch is built on top of the search engine Lucene and provides powerful methods to search, filter and interact with the indexed data.
3. Kibana, the official data visualization plug-in for Elasticsearch provides a user-friendly graphical interface to interact with the content indexed in the Elasticsearch database. It can be used to generate a variety of different visualizations and provides tools for extracting comprehensive statistics from the data.
4. Existing projects such as *Apache Metron* and *Moloch* (Section 1.2) have proven the suitability of these tools for network data analysis and forensic tasks.

4.4.1.1 Keywords

In order to make this Section easier to follow, we briefly introduce some of the most relevant Elasticsearch terminologies.

Index An index is a collection of documents that have somewhat similar characteristics. In this work for instance, we use a PCAP index, an index for Bro and a Snort index.

Document A document refers to a sample of data in JSON format, which is stored in an index. Each document has a type, an ID and contains zero or more **fields** (key-value pairs). For example, in our PCAP index, a document represents a single packet, in the Bro index a flow and in the Snort index an alert.

Mapping A mapping defines how a document should be mapped to the search engine. It includes a definition of its searchable fields and the corresponding datatypes [25].

Node A node is a running instance of Elasticsearch which belongs to a cluster.

Shard Very big indices may not fit on the disk of a single node or the database may be too slow to answer search requests from a single node alone. To solve this problem, Elasticsearch provides the ability to divide your index into multiple pieces called shards. The number of shards can be defined when creating a new index.

Replicas A replica is a copy of a shard, serving as back-up in case primary shard fails.

Bulk API The bulk API enables the execution of many index/delete operations in a single API call. This can greatly increase the indexing speed [25].

Dashboard A Kibana dashboard displays a collection of visualizations and searches for a particular index.

We used Elasticsearch and Kibana both in version 6.2, running on a single node with 10 Intel Xeon E5-2699 cores and 16GB RAM. For each index we set the number of shards to 1 with zero replicas. To load and index our data into Elasticsearch we wrote custom scripts using the official Python API [26] for Elasticsearch.

In the following we describe the process of creating an index for each of our three datasources (PCAP, Bro, Snort):

4.4.1.2 PCAP indexing

Indexing the complete PCAP data has proven to be the most challenging task when building up the database. In a first step, the raw PCAP data has to be converted into a form that is compatible with the indexing APIs of Elasticsearch. We used *TShark* [87], the terminal oriented version of the popular *Wireshark* packet analyser tool to convert the PCAP data into the human-readable JSON format which can be directly indexed into Elasticsearch. We used the following Tshark command for the PCAP-JSON conversion:

```
tshark -r packets.pcap -T ek -o tcp.desegment_tcp_streams:FALSE > packets.json
```

The *-T ek* option generates a newline delimited JSON format (ndjson) which is suitable for import into Elasticsearch using the Bulk API [87]. Note that Wireshark version 5.5 or newer is required, as older versions generate JSON where duplicate field names can occur, which is not compatible with Elasticsearch 6.0 and newer versions.

tcp.desegment_tcp_streams:FALSE is another critical option which deactivates Wiresharks TCP packet reassembly mechanism. If left active, Wireshark will attempt to reassemble files that have been split up into smaller chunks for transfer. This can be particularly problematic for big files as the maximum document size for bulk indexing in Elasticsearch is 2GB [25].

Wireshark can recognize more than 2000 protocols containing over 200.000 fields [89]. However, the majority of these fields will most likely never be searched on and since a large number of fields can slow down both indexing and query speed, it is not recommended to index all of these fields [89]. This can be achieved by setting the "dynamic" option in the corresponding mapping to *False*. Even so, all the non-indexed fields will still be stored in the Elasticsearch database but they will not be searchable nor aggregatable. In Appendix B, we include the exact mapping used for indexing the PCAP data.

4.4.1.3 Bro & Snort indexing

Bro and Snort data both produce human-readable output and can be configured to produce logs in Comma Separated Value (CSV) fileformat. While Bro by default generates tab-separated CSV logs, to do so in Snort the we used the *alert_CSV* output plugin [78]. We parse and load it into our database using custom python scripts.

4.4.1.4 Generating links between the indexes

After indexing all the data, another challenge is to create links between the different indexes in order to enable pivoting between the different traffic representations during the data analysis. For instance, when looking at a particular flow in the Bro index, the link should enable to switch to the PCAP index showing the packets belonging to the corresponding flow. To implement this feature, we make use of *scripted fields* [27]. Elasticsearch's scripting language *Painless* can be used to program fields to compute data on the fly using the data in the Elasticsearch indexes. Inside a scripted field we construct a URL by string concatenation containing the target index and a crafted query that will be executed when clicking on the link. Figure 4.3 shows two documents in the Bro index containing the described scripted fields. The "packets" links will direct the user to to the PCAP index and automatically issue a query matching the source and destination IP as well as the timestamp of the corresponding flow.

To make the queries between different indexes easier, we renamed the fields we used to match on, such that these field names are identical among our three indices. For instance we define the source IP address fieldname to be "src_ip". As in Bro this field by default is named "id.orig_p" we have to rename the field accordingly when indexing the data. For the Bro and Snort data we rename these entries in our custom python scripts. To rename the IP and and port fields of the PCAP data we deploy an ingestion node [24] in the database to pre-process the documents before the actual indexing happens.

	src_ip	dst_ip	src_port	dst_port	duration	proto	packets
00:02:54.893	10.7.2.45	224.0.0.252	63,434	5,355	0.106	udp	packets
00:02:54.999	10.7.3.254	10.7.3.156	45,264	445	0.001	tcp	packets

Figure 4.3: Two documents from the Bro index displayed in Kibana. The "packets" column contains scripted links to pivot from the Bro to the PCAP index.

4.4.1.5 Creating Kibana dashboards and visualizations

Once the data is indexed in Elasticsearch, the Kibana plug-in makes it easy to interact with the data and to generate custom visualizations. We create a Kibana dashboard for each of our three indices (PCAP, Bro, Snort). In the following we present two showcases where we detect DoS and SQL attacks in the LS18 dataset, using some the visualizations we created in Kibana.

DoS detection In this paragraph we go through a practical usecase where we identify a potential DoS attack in the LS18 data using some of our Kibana visualizations. Figure 4.4 shows a plot in our Bro index, illustrating the timeline of the number of connections per hour in the LS18 data. We observe a very dominant peak during the morning hours of the exercise's first day.

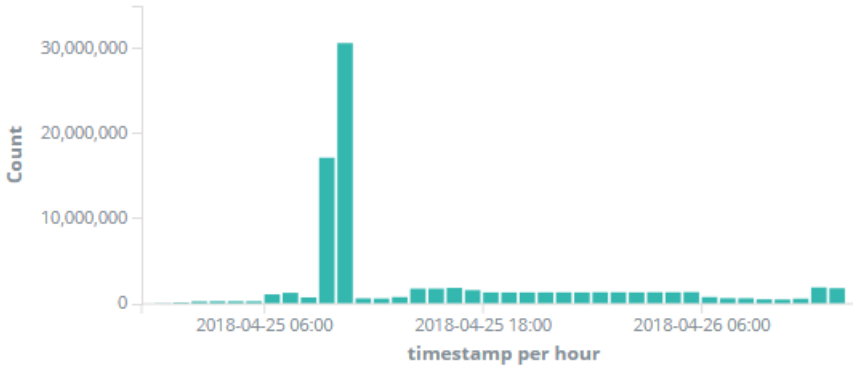


Figure 4.4: Kibana visualization in Bro index showing potential DoS attack in the LS18 data

Zooming into the affected time-window and counting the different source IP addresses that opened connections during that time reveals the IP addresses that were most active during this period (Figure 4.5).

Source IP	Count
10.16.2.6	28,014,944
10.16.8.180	14,118,872
151.216.16.5	2,726,188
151.216.16.36	1,603,602
10.16.8.151	259,858
10.16.2.136	155,566

Figure 4.5: Source IP counts of connections opened during time of the potential DoS attack

We observe that the IP addresses 10.16.2.6 and 10.16.8.180 were particularly active. Figure 4.6 shows some of the corresponding flows in more detail. We observe connections initiated by one of the suspicious IP addresses identified before, going to several different destinations using unusual destination ports. However, all of these flows have connection state S0. According to Bro's documentation (see Table 2.1), S0 describes flows where a connection attempt is seen but no reply is observed. So 10.16.2.6 is sending a vast amount of TCP SYN packets to different destinations, which matches the characteristics of a DoS SYN flood attack.

	src_ip	dst_ip	src_port	dst_port	duration	conn_state	proto
09:59:41.052	10.16.2.6	10.16.2.215	50,296	17,380	0	S0	tcp
09:59:41.052	10.16.2.6	10.16.2.108	50,296	11,040	0	S0	tcp
09:59:41.052	10.16.2.6	10.16.2.61	50,296	11,781	0	S0	tcp
09:59:41.052	10.16.2.6	10.16.2.158	50,296	8,470	0	S0	tcp
09:59:41.052	10.16.2.6	10.16.2.51	50,296	7,451	0	S0	tcp
09:59:41.052	10.16.2.6	10.16.2.137	50,296	14,797	0	S0	tcp
09:59:41.052	10.16.2.6	10.16.2.26	50,296	15,803	0	S0	tcp

Figure 4.6: List of some of the connections opened during the DoS attack

SQL injection Another practical usecase we show in the following is the detection of SQL injection attacks. Looking at the SQL response messages extracted by Bro's MySQL-analyser in Figure 4.7, we can easily identify the suspicious activities. The "access denied" messages indicate that somebody tried to access the admin and root accounts on the SQL server. With our setup we can easily identify the IP address that issued these requests as well as the target IP. We observe that the IP address 151.216.7.36 that made the requests belongs to a host in the internal network that was to be protected by the blue team. So we can conclude that at this point in time this machine was compromised and under the control of the red team.

MySQL Responses	Count
Access denied for user 'admin'@'151.216.7.36' (using password: YES)	58
Access denied for user 'root'@'151.216.7.36' (using password: YES)	1
Duplicate entry 'localhost-admin' for key 'PRIMARY'	1
There is no such grant defined for user 'root' on host 'localhost'	1
You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'uses www' at line 1	1

Figure 4.7: SQL responses from Bro's mysql.log

Chapter 5

Intrusion detection using machine-learning based techniques

In Section 4.3 we described how we obtained labels for the C&C sessions opened by compromised machines in the network.
In the first part of this chapter, we describe the supervised machine-learning based models we develop, that are able to predict such C&C sessions on previously unseen data. In Section 5.1 we define the network traffic features that we use to train our models and in Section 5.2 we explain the pre-processing methods that we apply to the data before training. We then describe in Section 5.3 how we select the best feature subset and the model-type that performs best in this classification task. We then conduct a thorough analysis of the robustness of the obtained models and identify possible attack vectors.

In the second part of this chapter, in Section 5.4 we investigate unsupervised clustering algorithms in order to identify malicious activities in the data without relying on prior information. We start the analysis using the novel CICIDS2017 dataset [73], where we have complete information about the malicious activities in the data. We then go back to the Locked Shields data where make a proof of concept showing the feasibility of applying unsupervised algorithms on this data by reporting high detection rates of the C&C sessions mentioned above.

5.1 Defining good network traffic features for IDS tasks

One of the main challenges in machine learning tasks is always to define adequate features to be used to train the models. For instance in a classification scenario, it is paramount that the distributions of the feature values differ for the different classes that are to be distinguished. If this condition does not hold, it is impossible for any model to differentiate between the different classes.

With a dataset consisting of network traffic captures, looking at packet contents only does not provide enough information to detect most of the possible attack types being used by hackers. Information about the context in which a particular packet occurs is necessary in most of the cases. For this reason, most investigations in machine-learning based intrusion detection calculate flow-based statistics rather than defining features on the packet-level.
The commonly used flow-based features can be categorized mainly into three different types [84]:

1. Basic features
Features that can be extracted by simple inspection of headers (e.g. port numbers, protocol, flag counts) or simple comparison operations (e.g. flow duration)

2. Content features
Features that require processing of the packet payloads. (e.g. Log-in success, Nr. of failed log-in attempts)

3. Traffic features

Features that describe time-based flow statistics. They can be further differentiated between features calculated over multiple flows inside a specified time-window, and time-based features that are calculated within single flows only (e.g. packet interarrival times).

In the following, we describe two different sets of features that have been proposed in past researches particularly for network intrusion detection tasks.

5.1.1 KDD-features

The dataset KDDCup99 and its updated and polished version NSL-KDD which we both described in Section 2.1.2 are among the most widely used datasets in intrusion detection research. A majority of the publications in this domain rely on these datasets and thus on the network traffic features they define. In Appendix C we provide a complete list of all 41 features used in these datasets, which we will refer to as KDD-features in the following.

Features 1-9 represent basic features, 10-22 account for content features and 23-41 describe time-based traffic features.

KDD feature-extraction tool

To extract the KDD-features from the raw LS17 and LS18 PCAP data, we used the feature extraction tool from [59]. The tool uses the same TCP state definitions as Bro, which we described in Table 2.1. Note that this tool does not calculate the content features 10-22 but only the basic and traffic features. However, the content features are rarely used, as calculating such features is particularly costly and will not provide meaningful information when analysing traffic using secure protocols with encrypted payloads. Furthermore [39] conducted a thorough feature selection analysis on the NSL-KDD dataset, concluding that all the content feature were either negligible or provided only little contribution to the intrusion detection task (see Figure 5.1). As the Locked Shields datasets in contrast to NSL-KDD contain also encrypted traffic, we expect these feature to be even less relevant on this data.

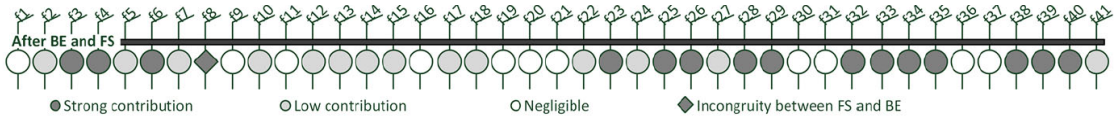


Figure 5.1: Classification of the feature relevance in NSL-KDD [39]

5.1.2 FlowMeter-features

In 2017, [73] proposed a new intrusion detection dataset CICIDS2017 (see Section 2.1.2), aiming to overcome the issues that accompanied older datasets such as KDDcup99. They presented a new and more extensive set of flow-based features. While using some of the previous common features, the authors focused on the definition of new time-related features. They propose features such as packet inter-arrival times, active times and idle times while always calculating the statistical min, max, mean and standard-deviation values. They rely on a bidirectional flow definition where the first packet determines the forward (source to destination) and backward (destination to source) directions. The statistical time-related features are also calculated separately in the forward and reverse direction [44].

In this work, we used primarily the features generated by FlowMeter rather than the KDD-features, as the performance of the obtained models using this feature set was significantly better (see Section 6.1.1).

A complete list of all the contained features can be found in Appendix C.

The FlowMeter tool

To generate this set of features for the Locked Shields data, we used a modified version of the *FlowMeter* feature extraction tool [33], provided by the researchers that have proposed this new feature-set [73].

FlowMeter is implemented in Java and comes with a simple GUI as depicted in Figure 5.2. In the GUI we can specify the directory holding the PCAP files to be processed ("Pcap dir"). The tool will generate a separate CSV file for each of the PCAPs found in the defined directory. If only a single PCAP file is to be processed, the tool will also accept the path to this particular file as input instead of a directory path. The generated CSV files holding the extracted flow statistics will be stored in the directory specified in the "Output dir" input textfield.

The TCP flows are terminated upon connection teardown when receiving a FIN packet or when reaching a timeout value. UDP flows on the other hand are always terminated when exceeding the flow timeout. The flow timeout value can be configured arbitrarily in the GUI (in microseconds). We have run test with several timeout values (300s, 120s, 15s) while observing no significant variations in the performance of our trained predictors described in Section 5.3. We therefore fix the timeout value to 15s which matches the optimal value found in [44]. A low timeout value furthermore has the advantage of reduced memory consumption during the feature extraction process, as opened connections have to be kept for a shorter time-period in memory. We found memory consumption particularly to be a problem for time-periods where particularly high network throughput occurs such as during the DoS attack in LS18 as described in Section 4.4.1.5. Using a lower timeout value, we can reduce the memory consumption significantly during this time period, preventing possible memory overflows.

In addition to the flow timeout value, an activity timeout value can be defined (default: 5 seconds). The tool differentiates between two possible flow states; active and idle. When no new packet matching the flow's 5-tuple properties is observed for a duration exceeding the activity timeout, the flow state will be set to idle. If a new packet is added to an idle flow before the flow timeout is reached, the flow state will be set to active again.

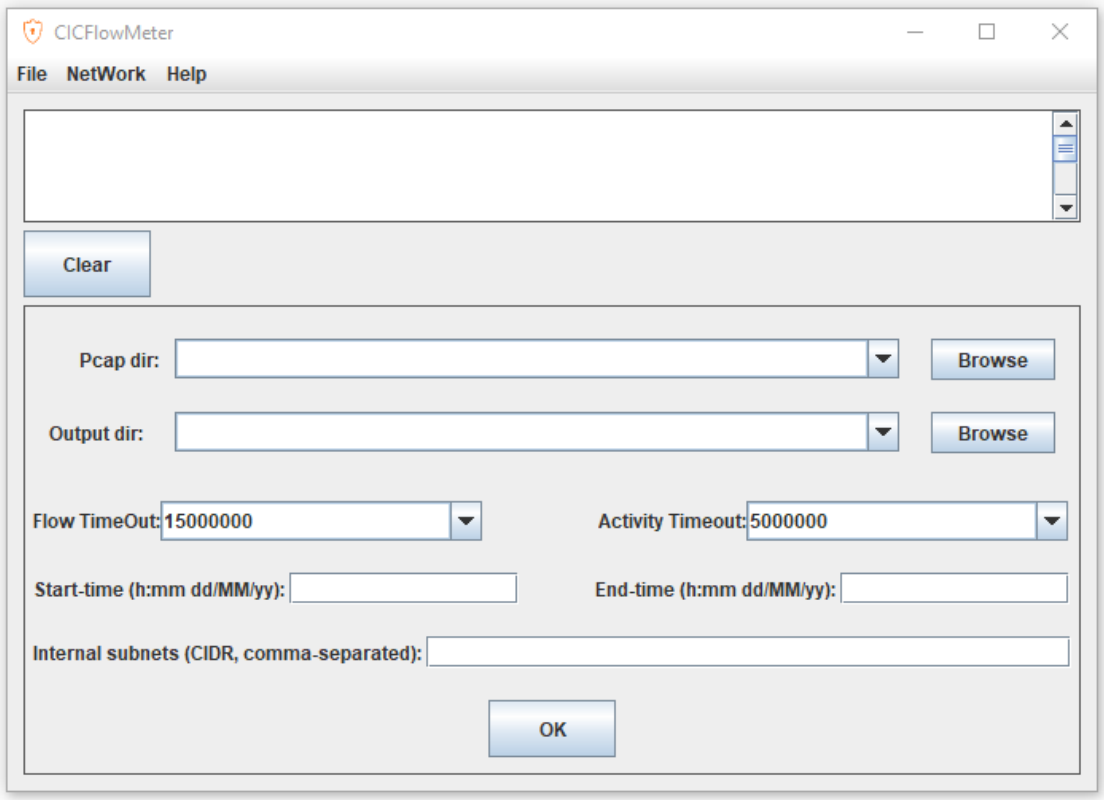


Figure 5.2: The GUI of the FlowMeter tool used for feature extraction

We have implemented several modifications to calculate additional features and also to correct

some of the shortcomings in the code. In the following we list the main modifications of the tool we have made:

1. Prevent memory overflows when processing big PCAPs
In the original version of the FlowMeter tool, the finished and timed-out flows were kept in a list in memory during the whole extraction process. Only after parsing the whole PCAP file the collected flow features would be flushed out to a CSV file holding all extracted features. While this is unproblematic when working with small PCAP files, in our case it lead to memory overflow errors when processing the LS17 and LS18 traffic captures. We modified the code to periodically flush the collected flows to the CSV file and deleting them from the list in order to free up memory.

2. Adding a new feature "Dst IntExt"
We added a new binary feature called "Dst IntExt" which has value 0 when the destination IP address of a flow belongs to an internal host and value 1 if the destination is an external host. We added a new input textfield to the GUI of the FlowMeter tool, where the user can define the subnet masks of the internal networks in CIDR notation (see Figure 5.2).

3. Time filter for PCAPs
We added two new text input fields "Start-time" and "End-time" (see Figure 5.2) to the FlowMeter GUI. When processing multiple PCAP files in the specified directory, only the PCAPs falling inside the specified time-window will be processed. Furthermore, the names of the already processed files in this directory will be added to a log file which prevents them be processed again when running the tool repeatedly in the same directory. When capturing network traffic using tools such as *tcpdump*, the process is often configured to generate new PCAP files after intercepting a certain amount of bytes or after a defined time-period, instead of writing all the traffic into one huge file. We introduced this feature to enable the user to only process a subset of the generated PCAP files inside the capturing directory within a time-period being of particular interest.

4. Filter out flows with TCP SYN count = 0
We filter out the extracted TCP flows where no SYN packets were observed. The motivation for doing so is explained in Section 5.2.1.

Note that besides providing offline PCAP processing, FlowMeter includes a real-time feature-extraction mode, where the tool listens to the traffic on the network interface. However, in this thesis we did not test this feature. We sincerely doubt that the tool could handle the high throughput of the network traffic in the Locked Shield exercises in real-time.

5.2 Data preprocessing

In this section, we describe the preprocessing steps that we have applied to the data before training our models. Filtering out some unwanted samples such as outliers or performing mathematical transformations on the feature vectors is a crucial step in many machine-learning tasks.

5.2.1 Filtering out undesired flow entries

When working with the features as generated by the two tools presented in Section 5.1 (KDD feature-extractor, FlowMeter), we observed that both tools lead to some extracted flows in the output that are not meaningful to use in our task.

KDD
In Section 4.2 we explained the reason for the occurrence of duplicate packets in the LS17 and LS18 PCAP data. When using the KDD feature-extractor we observed that in some cases these duplicates lead to redundant and not meaningful flow entries. For instance, a duplicate FIN ACK packet in a TCP connection will lead to a new flow entry with OTH connection state (see Table 2.1), as after observing the first FIN ACK packet the tool already closed the flow. The same issue however occurs also in case of retransmitted instead of duplicate packets.

Based on this observation we decided to use only the extracted flow entries with TCP connection states that indicate normal establishment of the TCP connection. Following the state definitions as listed in Table 2.1, S1, S1, S3, SF, RSTO and RSTR represent the states of connections that have been established. By limiting our flows to these states, we can safely filter out the undesired flow entries described above. As we are aiming to predict C&C sessions, where generally a connection has to be successfully established between the compromised machine and a server under control of the attacker, focusing on flows with established states only will not limit the quality of the resulting models trained for this particular classification task.

FlowMeter
In the CSV file holding the flows extracted by the FlowMeter tool, we identified a significant amount of undesired TCP flow entries. The main issue is that FlowMeter uses a very naive logic to track TCP connection states and particularly for terminating TCP connections. A TCP connection is terminated after receiving the first FIN packet matching the corresponding flow's 5-tuple properties. However, usually both parties will send FIN and the corresponding ACK packets. So after receiving the first FIN packet, FlowMeter will close the flow and then open a new flow when processing the following ACK packet. When receiving the other FIN packet the tool will close the second flow and then generate a third flow when processing the second ACK packet. So in this scenario, only one of three extracted flows holds meaningful information.

To overcome this issue, we filter out TCP flows where no SYN packet has been observed. Doing so, we keep only the first and valid of the three extracted entries above. We implemented this filtering step directly in the FlowMeter tool, so our modified version of the tool will automatically filter out the invalid flow entries.

5.2.2 Standardization

The features of our data rely on different measurement units and the variances of the feature vectors thus may strongly vary. In Section 2.5.4 we introduced the standardization preprocessing process and explained that for some machine-learning algorithms it is crucial to bring all features first to the same scale before training. We applied standardization to the training- and test-sets for all machine-learning models we evaluated, except the random forest models, as they are invariant to data scaling.

5.2.3 Class balancing

The two classes (normal, malicious) in our dataset are fairly unbalanced. Many machine-learning algorithms do not perform well when trained on unbalanced datasets.
Table 5.1 shows the percentages of the malicious class samples in the LS17 and LS18 datasets generated with the FlowMeter and the KDD feature-extraction tools. The slight divergence between the percentages when using the KDD feature-extractor versus the FlowMeter tool originates from the different conceptions of flows and connection states as well as the different filtering methods applied (see Section 5.2.1).

	Host-labelling		**Session-labelling**	
	LS17	**LS18**	**LS17**	**LS18**
KDD	15.01%	6.11%	0.3%	-
FlowMeter	12.02%	9.99%	1.1%	-

Table 5.1: Percentages of the malicious class samples in LS17 and LS18 using the KDD and the FlowMeter feature-extraction tools.

One method to balance the different classes is to subsample from the more frequent class until we have the same amount of samples for both classes. In the experiments we conducted using supervised learning, we evaluated the performance of all our models with and without the class-balancing preprocessing step. In the unsupervised domain on the other hand, class-balancing cannot be applied as there we assume that there is no prior knowledge available about the classes in the data.

5.2.4 Encoding of categorical features

In Section 2.5.5 we explained that not all machine-learning algorithms can deal with categorical features by default. While the FlowMeter-features only contain numerical features, the KDD feature-set contains three categorical features (Nr. 2-4 in Table C.1).

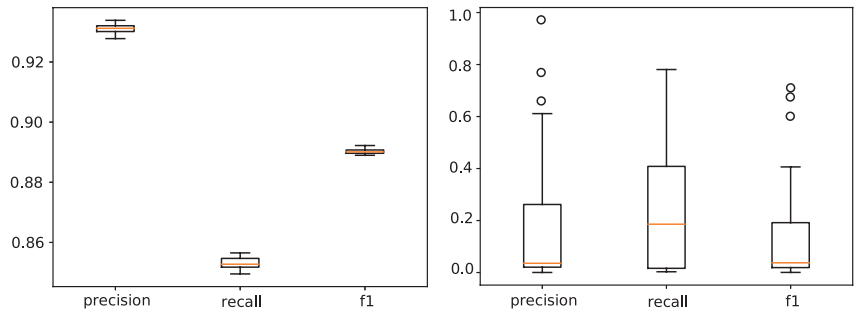
In our experiments, we apply one-hot encoding as described above for training model-types that cannot work with categorical values directly.

5.2.5 Generating meaningful training and validation sets

In Section 2.5.2, we introduced common approaches to split the dataset into train, test and validation sets in order to counteract overfitting of the final models on the train data.
We first tested a repeated K-fold cross-validation scheme on the LS17 data where the samples are shuffled after each iteration. While this is a common practice in machine-learning, we were concerned that this would lead to train and validation splits that correlate strongly with each other, making the classification task too easy. An intuitive proof for this hypothesis is that due to the shuffling process, the train and validation sets both will contain samples corresponding to the same C&C sessions in the data. Those samples possibly correlate strongly with each other which facilitates the classification task significantly. However, in a real-case scenario we want to be able to also detect sessions that have not been observed before. Hence, the training-set to fit our model on, should contain samples of C&C sessions between different source-destination IP pairs than the validation set. To ensure this, we randomly split the list of malicious IPs belonging to the listener servers for the C&C sessions (see Section 4.3), into two parts A and B. In a second step we assign all malicious samples corresponding to IP addresses that are in list A to the training set, while we use the list B for selecting the malicious samples we include in the test set. We repeat this process ten times, obtaining ten different pairs of training and validation sets that do not contain flows of the same C&C sessions.

In Figs. 5.3 and 5.4 we compare the performance of a random forest classifier using a repeated K-fold cross-validation scheme with the performance on the specially crafted training and validation sets as described above for both session-labelled and host-labelled data (see Section 4.3). Note that we train and evaluate the models for all of the ten dataset-splits in both directions, so in total we train and evaluate twenty models on twenty different validation sets. We use the best 20 FlowMeter-features found by the feature selection process described in Section 5.3.2 while balancing the classes of the training sets. We observe as expected that the achieved performance scores using cross-validation are significantly higher while variances are much smaller. While for the host-labelling we achieve high scores for both experiments, using the sessions-labelling, performance drops significantly on the datasets splitted by IPs.

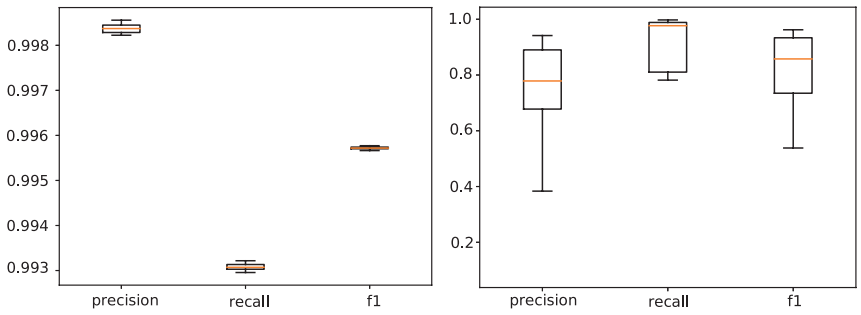
Based on these observations we prefer evaluation on the train and validation sets splitted by IP rather than using cross-validation, as this makes the classification task more challenging and realistic.



(a) Random 2-fold Cross-validation, repeated 10 times

(b) Data randomly splitted by IPs, repeated 10 times

Figure 5.3: Performance comparison of a random forest model using random cross-validation (left) and train and validation sets splitted by IPs for "session-labelling" (right).



(a) Random 2-fold Cross-validation, repeated 10 times

(b) Data randomly splitted by IPs, repeated 10 times

Figure 5.4: Performance comparison of a random forest model using random cross-validation (left) and train and validation sets splitted by IPs for "host-labelling" (right).

5.3 Supervised detection of C&C sessions

In this Section, we describe how we develop supervised machine-learning models to predict the malicious C&C sessions we have labelled in the LS17 and LS18 datasets.

5.3.1 Selecting the performance measures for model evaluation

When evaluating the performance of machine-learning models it is crucial to define performance measures that are suitable for the task to be solved. **Accuracy** is one of the measures most commonly used in the domain to asses the prediction quality of a model. However, the LS17 and LS18 datasets are both fairly unbalanced (see Section 5.2.3). For instance, only 1.1% of the samples in the LS17 dataset labelled by sessions belong to the malicious class. If we use a predictor that classifies all samples as normal, we would still get an accuracy score of 98.9%, which is misleading as the model completely fails in the classification task.

Precision and **Recall** are two performance measures that are commonly reported together and are more suitable for evaluating predictions on unbalanced datasets. The **F1** represents a combination of the two (harmonic mean value). They are defined as follows:

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN} \qquad F1 = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} \tag{5.1}$$

TP = Number of true positives
FP = Number of false positives
FN = Number of false negative

In our classification task, we define the malicious class to be the positive class. More intuitively, precision describes the fraction of predicted malicious samples that are actually malicious. It can be thought of as a measure of a classifiers exactness. A low precision often indicates a large number of False Positives.
Recall on the other hand describes the fraction of all attacks that have been correctly predicted. So if recall is low, this means that the model failed to detect many of the malicious samples.

While in the intrusion detection domain, both measures are meaningful, we weight the precision score higher than Recall for our classification task, as a high number of false positive would mislead security analysts during their operation. To make a numerical example, an IDS tool with 99% precision and 60% percent recall is strongly preferred to a tool where the scores are the other way around.

5.3.2 Feature-selection

In this part, we present the feature-selection strategy we followed to select an optimal subset of the features to train our models.

There are several reasons why going through a thorough feature-selection process before training the final predictor models is of high importance:

1. Removing irrelevant as well as high correlating features from the data might improve the prediction performance significantly.

2. Reducing the number of features and thus the dimensionality of the dataset can strongly reduce training and inference times, resulting in more cost-effective models.

3. The lower feature count increases the overall interpretability of the problem, leading to a better understanding of the underlying process.

1. Feature elimination
In a first step, we remove features that are constant or always zero in all samples. Such features do not contain any information relevant to the classification task and thus it is not meaningful to include them into the training data.

After that, we calculate the correlation between each feature vector and the labels. The aim is to eliminate features with zero or very little correlation to the labels. As the popular Pearson correlation coefficient [11] can only capture linear relationships between two vectors, in addition we calculate the Mutual Information (MI) metric [10] which also measures non-linear relationships. We then only eliminate features where both measures are particularly small or close to zero.

Figure 5.5 shows the 25 FlowMeter-features with the lowest correlation (absolute values) and MI scores while Figure 5.6 illustrates the same for all KDD-features, excluding the categorical features. Table 5.2 lists the FlowMeter features that we removed in these two steps while Table 5.3 lists the eliminated KDD-features. In total we are able to reduce the number of FlowMeter features from 75 to 58 by removing 17 of the features while we removed 7 features of the total 28 KDD-features.

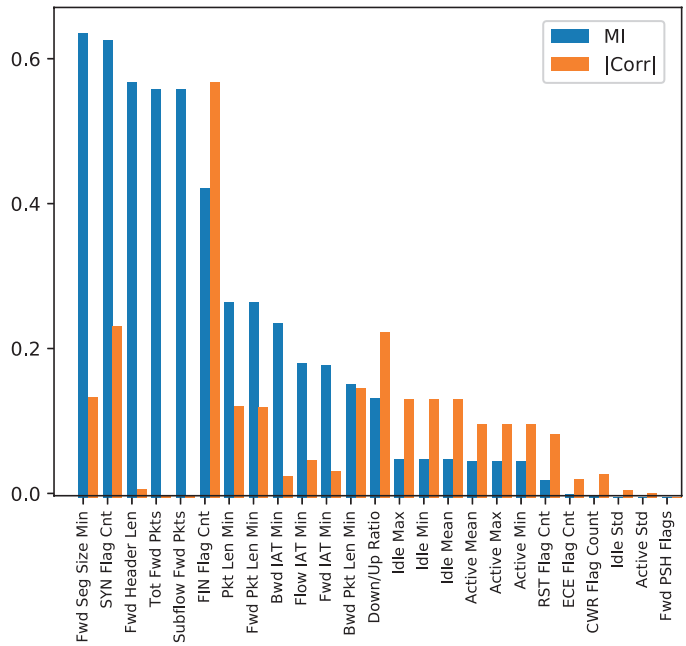


Figure 5.5: The 25 FlowMeter-features with lowest MI and Pearson correlation scores

2. Recursive feature elimination
Machine learning models, such as decision trees or forests calculate internally an inherent ranking of the features fed into the model. For other models such as SVM or Lasso it is possible to calculate such a ranking from the structure of the model. For instance, big weight values might indicate that a feature is particularly important.

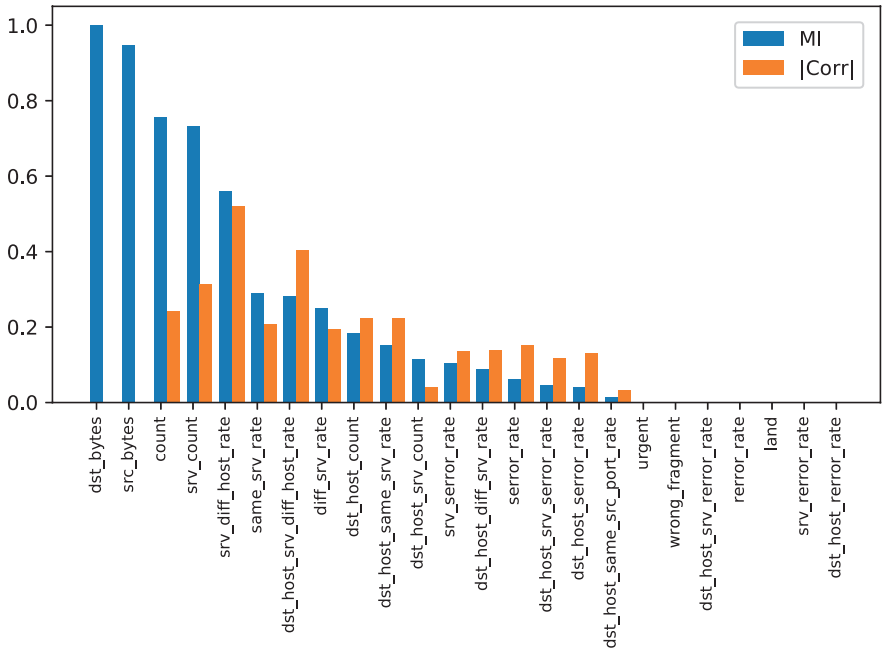


Figure 5.6: The MI and Pearson correlation for the KDD-features, excluding the categorical features.

Constant values	Low correlation/MI
Bwd PSH Flags	Fwd PSH Flags
Fwd URG Flags	CWR Flag Count
Bwd URG Flags	Active Std
URG Flag Cnt	Idle Std
Fwd Byts/b Avg	Flow IAT Min
Fwd Pkts/b Avg	Fwd IAT Min
Fwd Blk Rate Avg	Bwd IAT Min
Bwd Byts/b Avg	
Bwd Pkts/b Avg	
Bwd Blk Rate Avg	

Table 5.2: Removed FlowMeter-features in the two feature elimination steps

We have chosen a feature-selection scheme that is based on random forest's importance scores for the following reasons:

1. Random forest achieves good performance on a variety of datasets with little or no parameter tuning.
2. Random forest can deal with unbalanced datasets.
3. The classifier performance of random forests is invariant to strongly correlating features.
4. Linear computational complexity and parallelizable calculation.
5. It can deal with categorical features.

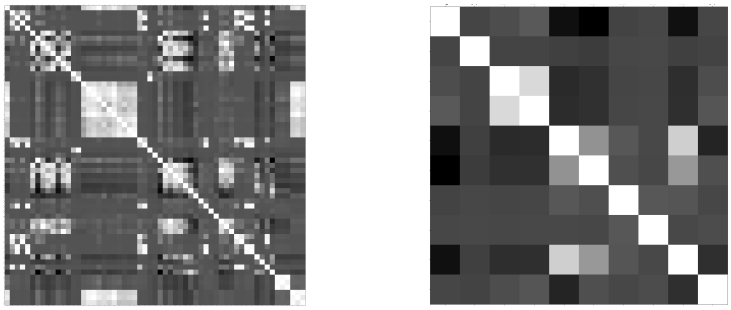
We use sklearn's *feature_importances_* parameter of the *RandomForestClassifier* class [75] to obtain a score value for each feature after training the model. This score is commonly referred as "Gini importance" or "mean decrease impurity" [50].

To reduce the number of features, we apply a recursive feature elimination scheme, removing the feature with the lowest score in each iteration. Eliminating the features one by one is crucial,

Constant values	Low correlation/MI
-	dst_host_rerror_rate
	dst_host_srv_rerror_rate
	wrong_fragment
	srv_rerror_rate
	urgent
	land
	rerror_rate

Table 5.3: Removed KDD-features in the two feature elimination steps

as importance scores in random forest can spread over multiple features with redundant information. So if we train the model with several strongly correlating features, the calculated scores might be all relatively low, also if keeping the information contained by these features might be relevant. By removing only one feature at once, the scores of the remaining correlating features increases, and if we repeat that, we finally end up with only one of the redundant features which then automatically gets assigned a higher score.
Figure 5.7 shows the self-covariance matrices of the FlowMeter-features for all 58 features and the subset of the best 10 features obtained by our feature-selection process. We can clearly observe that while there are several strongly correlating features in the full feature set, the fraction of redundant features decreases tremendously when looking at the top-10 feature subset. This proofs that our feature selection algorithm succeeds in removing most of the correlating features from the data.



(a) Self-correlation matrix of all FlowMeter-features

(b) Self-correlation matrix of the top-10 FlowMeter-features

Figure 5.7: Feature self-correlation matrices. White means full correlation while black represents zero correlation between two features.

Another reason why eliminating only one feature per iteration is important, is that the feature importance scores can sometimes be very unevenly distributed. The highest ranked feature might achieve a very high and dominant score while the scores of the other features may be significantly smaller.

We summarize that small importance scores do not necessarily imply that the corresponding features are not important. However, using recursive feature elimination we can prevent the elimination of features that are relevant despite having scores much lower than the top-scored features.

Table C.4 in the appendix lists the FlowMeter- and KDD-features ordered by importance, excluding the constant or zero value features as obtained in the first step of our feature selection scheme.

FlowMeter vs. KDD In our experiments we found the quality of the predictions by the models trained with the FlowMeter-features to be significantly better than models that rely on the KDD feature-set (see Section 6.1.1.3). Due to this finding, we base the following analysis and the development of the final model mainly on the FlowMeter-features rather than KDD.

5.3.3 Model selection

In the following, we describe the strategy we followed to select the model-types that fit best to our data and the selected classification task.

We are given the two datasets LS17 and LS18 which have been acquired completely independent of each other.

Step-1 Using the LS17 data, we evaluate several supervised models on the specially crafted training and validation splits as described in Section 5.2.5 to find the models that work best on this data.

Step-2 We select the best performing models, retrain them on the complete LS17 data and then evaluate them on the previously unseen LS18 data. We then go the other direction, training the selected models on the LS18 data and the evaluating the predictions on LS17.

Following this strategy we ensure to overcome possible overfitting issues, as on one hand we apply a cross-validation inspired scheme in step-1 while on the other hand in step-2, we evaluate the trained models on previously unseen test data that was acquired independently from the training data.

We run the presented model selection strategy using five popular supervised models; Support Vector Machine (SVM), Logistic Regression, Naive Bayes, K-Nearest Neighbors (K-NN) and Random Forest (RF).
We found that a random forest classifier trained with the top-10 FlowMeter features works best on our data (see Section 6.1.1).

Session-labelling vs. host-labelling Note that while we have labelled the LS17 data both using the session-labelling and the host-labelling (see Section 4.3), we could only obtain the host-labelling for the LS18 dataset. So for the session-labelled data we only evaluate step-1 of our model selection strategy.
However, after a thorough analysis of both labelling strategies, we decide to train our final models using the host-labelling only for the following reasons:

1. The performance of the predictors using the session-labelled data was not satisfactory with the tested models (see Section 5.2.5).
2. The malicious samples using the session-labelling represent a subset of the malicious samples in the host-labelled dataset. The "host"-model is thus also capable to classify these samples.
3. We could only label the LS17 dataset with the session-labelling strategy. However, as the goal is to use the final model in future challenges, the evaluation on the LS18 data is crucial.
4. The classes in the dataset using the session-labelling are far more unbalanced in comparison to the host-labelled dataset (see Table 5.1), making the training of good classifiers more difficult.
5. Hypothesis 1 as derived in Section 4.3 suggests that both the session-labelling and the host-labelling mainly flag samples corresponding to C&C sessions controlled by the red team.

5.3.4 Increasing the robustness of the model

Following the presented model-selection strategy in Section 5.3.3 we found that a random forest model with the top-10 FlowMeter-features obtained by our feature-selection algorithm performed best on our classification task.
In this Section we conduct a thorough analysis of the robustness of this model. First, we generate adversarial inputs simulating an attacker aiming to avoid detection of his opened C&C sessions to a compromised host. In a second experiment, we simulate packet loss that might occur when the PCAP capturing infrastructure gets overloaded.

In this process we identify methods to improve the overall robustness of the model and in particular its resilience to attacks.

5.3.4.1 Simulating attacks using adversarial inputs

In the following, we identify possible attack vectors of the selected random forest model. We assume a white-box scenario, where the attacker has full knowledge about the model we deploy and the features it relies on.

Changing the appearance of the C&C sessions using Cobalt Strike
As our model detects C&C sessions opened within the Cobalt Strike framework, we first focus on the options and parameters that Cobalt Strike provides to alter the appearance of these sessions.

Two of the main parameters that can be chosen in Cobalt Strike and that are used by the red team during the exercise are the **sleep-period** and the **jitter** of a C&C session. The sleep-period defines the time period of the compromised machine contacting the C&C server to see if the attacker has issued new commands to be executed. In other words, the compromised machine opens each sleep-period seconds a connection to the C&C server. The jitter configures the deviation from this periodicity. A jitter value of 20 % means that the C&C session will vary its sleep time by up to 20%, making the sessions harder to detect as they don't occur with an exact period anymore. The FlowMeter-features we use are invariant to both these parameters, as they focus on timing statistics within single connections and do not depend on the time elapsed between the periodic connections corresponding to the C&C session.

Cobalt Strike's **Malleable C2** tool [49] allows to customly design the headers of the HTTP requests exchanged by the compromised host and the C&C server in order to vary the appearance of the different sessions. The motivation is to evade the detection of all other sessions if one session gets caught. However, since our model does not rely on content based features extracted by processing HTTP headers, changing their appearance will not lower the prediction quality.

Finding possible attack vectors
In the last paragraph we concluded that our model is invariant to changing the C&C session parameters that Cobalt Strike provides. In the following, we look at each one of our 10 selected features separately and propose how it can be attacked. For an explanation of each of the features please refer to Table C.2 in the appendix.

1. **Active Mean**
   The attacker could periodically inject packets with a period smaller than the activity timeout value configured in the FlowMeter tool during feature-extraction (see Section 5.1.2). Doing so, the connections corresponding to the C&C sessions would always remain in the active state, which will increase the active mean time.

2. **SYN Flag Cnt**
   The attacker can increase the count of packets with SYN flag set by injecting additional SYN packets in the forward direction (from compromised host to C&C server).

3. **Init Fwd Win Byts**
   This feature represents the TCP-window size that was set in the first packet sent by the compromised host. One possibility to alter this feature is to change the default TCP-window size in the operating system of the compromised machine.

4. **FIN Flag Cnt**
   This feature cannot be attacked by injecting additional FIN packets, as the FlowMeter tool will automatically close the flow upon reception of the first FIN packet.

5. **Bwd Pkt Len Min**
   The attacker could inject some particularly small packets in the backward direction (from C&C server to compromised host) in order to lower the min value.

6. **Flow Pkts/s**
   The attacker could inject additional packets in both or either one of the directions to increase the measured paket-rate.
7. **PSH Flag Cnt**
   Analog to the attack explained for SYN Flag Cnt.
8. **Fwd IAT Max**
   The attacker could inject additional packets to alter the IAT statistics
9. **Flow IAT Mean**
   Analog to the attack explained for Fwd IAT Max.
10. **Tot Fwd Pkts**
    The attacker could inject additional packets in the forward direction to increase the count of total pakets sent from compromised host to the C&C server.

We observe that most of the feature values can be attacked by injecting additional packets, either in the forward or the backward direction or both. This type of attack is possible, because the FlowMeter tool does not implement any form of sequence number checking. So the attacker can inject random packets matching the 5-tuple identifier of a connection corresponding to the C&C session while using invalid sequence numbers. While the receiver will drop such packets, FlowMeter will process them and take them into account for calculating the flows' statistics.
To execute such an attack, the threat actor would need to write custom scripts, that match the connection 5-tuple of each connection corresponding to the opened C&C session, and then injecting packets specially crafted to attack one or more specific features.
However, by implementing sequence number checking in the FlowMeter tool this type of attacks could be prevented (see Section 7.2).

Analysis of the sensitivity to the selected features
After identifying the possible attack vectors, what remains now is to test how sensitive our model reacts when changing the corresponding feature values.

In a first step we train several new models, where we exclude one of the ten features in each of these models. We then look at the decreases in precision and recall scores to identify the features that the model relies on most strongly. We excluded both of the IAT features together, as they contain redundant information.
Table 5.4 shows the precision and recall values of the nine resulting models after training on the LS17 set and evaluating on LS18.

Excluded feature	Precision	Recall
-	0.98	0.98
Active Mean	0.98	0.98
SYN Flag Cnt	**0.89**	0.98
Init Fwd Win Byts	0.99	**0.73**
FIN Flag Cnt	0.97	0.98
Bwd Pkt Len Min	0.98	0.98
Flow Pkts/s	0.98	0.98
PSH Flag Cnt	0.96	0.98
Fwd IAT Max, Flow IAT Mean	0.96	1
Tot Fwd Pkts	0.98	0.98

Table 5.4: Pecision and recall values of the nine resulting models where we excluded one of the top-10 features in each model. Training set: LS17, Test set: LS18

We observe that the model performance hardly decreases after excluding the features separately. This is a good sign as it means that our model does not rely strongly on each one of the ten selected features alone. The strongest decrease in performance scores occurs when leaving out the SYN Flag Cnt and the Init Fwd Win Byts features. Even so, the precision of the model would be still high enough to deploy the model and to successfully detect a majority of the C&C sessions.

Analysis of the SYN Flag Cnt feature In the previous paragraph we identified the SYN Flag Cnt to be particularly relevant to the model. Intuitively, it is not obvious why the count of the SYN flags in a connection is so important in our classification task. One information this feature holds is about the protocol. We know that if we observe SYN flags in a flow, the transport layer protocol has to be TCP. However, training the model without SYN Flag Cnt feature but with a protocol feature instead did not improve the prediction quality notably.

Next, we look more in detail at the actual value distribution of the SYN counts for the normal and malicious class samples. We find that while 4 is a common value for the normal class, it is not for the malicious class. Looking at the flows corresponding to value 4, we observe the following packets with the SYN flag set and thus increasing the counter:
SYN, SYN (dup), SYN-ACK, SYN-ACK (dup)
Note that (dup) refers to duplicate packet that occur if the packet traverses two VLANs in our internal network that are both configured for packet capturing such as described in Section 2.2. In flows of the malicious class, one or both of the duplicate ACK packets is missing, indicating that this packet only went through one VLAN. This is particularly common for hosts in external networks, as packets coming from the outside often pass only one VLAN to get to the target in the internal network.

We conclude that the SYN feature holds information about a host being either internal or external. This information indeed is relevant to our classification task, as the C&C servers responding to the compromised machines in our network are mostly external hosts. We introduce a new binary feature **Dst IntExt** that has value 0 if the responder IP address of a flow lies inside our network, and value 1 if this address belongs to an external host. Training a new model replacing the SYN Flag Cnt and the PSH Flag Cnt with the newly introduced Dst IntExt and Protocol features gives us the same performance as the model that relies on the SYN Flag Cnt feature. Note that we also removed the PSH Flag Cnt, as this feature, being also a flag count value might contain redundant information. An additional benefit next to the improved interpretability of our model is that we loose one of the attack vectors, as the attacker now cannot alter this feature anymore by injecting additional SYN packets into the flow.

Table 5.5 lists the precision and recall scores of three random forest predictors. The first model is trained with the original configuration (top-10 FlowMeter-features), the second model was trained without the SYN/PSH flag count features and the third was trained after replacing the SYN/PSH counts by the newly introduced Dst IntExt and Protocol features. We observe that by introducing the new features the model is able to fully recover from the losses introduced by removing the flag count features.

Note that we did not exclude the FIN Flag Cnt feature. The reason is that this feature for all samples either has value 0 or 1. It will have value 0 for flows using protocols other than TCP, or for timed-out TCP flows. So this feature does not depend on the duplicate packets as the other Flag counts, but does hold information about protocol and timeout.

	Precision	**Recall**
Original configuration	0.98	0.98
No SYN/PSH Flag counts	0.83	0.97
With Dst IntExt and Protocol	0.98	0.98

Table 5.5: Performance comparison of a Random Forest model trained with the top-10 FlowMeter-features (original model) to models without SYN/PSH count features, and a model relying on the new Dst IntExt and Protocol features

Simulating attacks by generating adversarial inputs
Next, we generate adversarial inputs crafted to attack each of the features of our model. We first look at the distribution of each of the ten features for both the normal class and the malicious class. Then we select values that are common for the normal class while being rarely observed in the malicious class instances. To simulate an attack of a particular feature, we replace the feature values in the malicious samples with values randomly subsampled from the values we have identified to be common for the normal class but not for the malicious class. The aim is to

make the appearance of the malicious samples more similar to the characteristics of samples belonging to the normal class.

The prediction scores of the models under attack are listed in Table 6.7.

Parameter-tuning to increase the robustness
So far, we have used a random forest ensemble model with ten fully expanded trees. We observe that the resulting trees in this configuration are fairly big, with node counts around 30'000 for the model trained on LS17, and 70'000 for the LS18-model. A common issue in decision trees is that if they grow too large, the risk of overfitting on the training data and poor generalization to new samples arises. Despite the recommendation of the random forest inventors [14] to fully expand the trees, we found that setting a constraint for the maximal depth of the trees increased the overall robustness of our model significantly. Restricting the depth of the individual trees to 10 reduces the node counts of the models to around 700 for the LS17 models and 900 for the LS18 models. However, when setting the constraint even lower, precision and recall values start to decrease.

Moreover, we found that the robustness can be further increased by increasing the number of trees in the ensemble. However, more trees also mean more computational cost and after a certain number of trees, the improvement is negligible. [62] conducted an evaluation with varying tree count values on 29 different dataset. They found that after 128 of trees there is no significant improvement in accuracy. Accordingly, we choose to increase the tree count from 10 to 128, which increases the resilience of our model to the simulated attacks even further.

Finally, by increasing the number of features used for training, the robustness of the model can be increased even more. In our final configuration, we raised the number of features from 10 to 20.

To recapitulate our findings we briefly summarize the three measures we found to help increasing the robustness of our model:

1. **Cutting back the trees by setting maximum tree depth constraint of 10**
2. **Increasing the number of trees in the random forest ensemble from 10 to 128**
3. **Increasing the number of features from 10 to 20**

The precision and recall measures before and after tuning the mentioned parameters of the model are listed in Table 6.7.

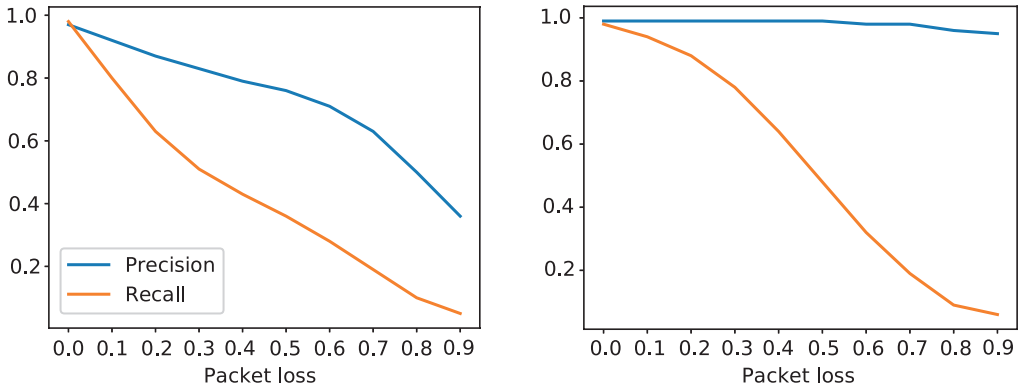
5.3.4.2 Simulating packet loss

In addition to driving the model with adversarial inputs, we analyse the impact of packet loss occurring during the capturing process on the prediction quality of our model to further evaluate the robustness. The goal is to simulate a situation during the Locked Shield exercise, where throughput on the wire is too high for the PCAP capturing system to handle. We measure the performance of the model with packet-losses from 10% to 90%, in 10% steps. To set up this simulation, we parse the PCAP file of LS18 and drop each packet with a probability of the configured packet-loss while generating a new PCAP file containing only the remaining packets. In a second step, we feed the generated PCAP files into the FlowMeter feature extraction tool. We then use our models trained on the complete LS17 dataset and perform predictions on the newly generated LS18 datasets where we simulated the packet-losses.

Note that we can also interpret packet-loss as a kind of attack. Remember that most of our feature statistics can be altered by injecting additional packets. Dropping packets has a similar effect on the flow statistics, but in the opposite direction.

Figure 5.8 shows the precision and recall values of two random forest models trained with 20 features each. The first model was trained using the original configuration we obtained before implementing the suggested changes to increase the robustness as outlined in Section 5.3.4. In the second model on the other hand, we replaced the SYN Flag Cnt and the PSH Flag Cnt with the Dst IntExt and Protocol features, increased the tree count to 128 and cut the tree

back to a maximal depth of ten. We observe that the tuned model is much more resilient to the packet-losses than the original model. While the original model is already down at 80% recall for a packet-loss of only 10%, the tuned model crosses the 80% mark at a packet-loss of 30%. Looking at the precision curves we note that while for the original model precision drops monotonously, the precision score stays fairly constant in the tuned version. This means that although the tuned model misses more and more of the malicious flows with increasing packet loss, the predictions the model makes are still precise and few false positives occur.



(a) Original model configuration including the SYN Flag Cnt and the PSH Flag Cnt features

(b) Tuned model with the new Dst IntExt and Protocol features, 128 trees, max_depth=10

Figure 5.8: Precision and recall values of the original and the tuned random forest models trained with 20 features on LS17 and evaluated under the simulation of packet-loss on LS18

5.3.5 Truncating packets during capturing

Traffic capturing using tools such as *tcpdump* can be a resource-intensive process if the throughput on the wire is high. To decrease the needed resources as well as the required diskspace for the recorded pcap data, *tcpdump* offers the option to truncate the recorded packets by setting the *snaplen* parameter [34]. In other words, when enabling this option, *tcpdump* will only store the first *snaplen* bytes of each packet.

We simulated this process by applying truncation with a *snaplen* of 96 bytes to the LS17 and LS18 pcap files. We then extracted the FlowMeter-features from these truncated versions of the datasets, retrained and evaluated our random forest classifiers on this data.

Table 5.6 shows the sizes of the original and the truncated versions of the PCAPs. We observe, that the sizes of the truncated PCAPs are reduced to only a fourth of the original size.

In Table 5.7 we report the precision and recall metrics for the tuned configuration of the random forest classifier as described in Section 5.3.4. We observe that the model trained and evaluated on the truncated data even performs slightly better.

PCAP	Original	Truncated
LS17	114GB	28GB
LS18	216GB	52GB

Table 5.6: PCAP sizes of the original and the truncated PCAPs where only the first 96 bytes of each packet are kept.

5.3.6 Deployment of the model as Intrusion Detection System (IDS)

Up to this point, we have developed a random forest based model that can detect flows belonging to Cobalt Strike's C&C sessions with high precision. However, while classifying the flows is important, processing this information directly might be challenging for the blue team.

	Original	Truncated
RF-LS17 (20F) - Tuned	0.99/0.97	0.99/0.98
RF-LS18 (20F) - Tuned	0.98/0.84	0.99/0.90

Table 5.7: Precision/Recall metrics for the tuned random forest classifier on the original LS17 and LS18 datasets, and the truncated versions where only the first 96 bytes of each packets are kept.

To deploy the model more effectively, we propose the use of a second inference stage after the flow classification process. In this stage, we accumulate the flows that have been classified as malicious, by their destination IP address. Furthermore, we store all source IP addresses of hosts that connected to these destinations while counting the number of connections matching the corresponding source-destination IP pairs. Finally, we order the accumulated list by the most prominent destination IPs in descending order. The top entries in this list now hold the destination IP addresses that occur most often in flows classified as malicious. Thus, the probability of these IP addresses for belonging to malicious C&C servers is particularly high, while the hosts with the source IPs that connected to these servers are likely to be compromised.

Deployment
Figure 5.9 illustrates the architecture of the IDS as described above. In the first stage, the traffic on the wire has to be recorded in PCAP format using tools such as *tcpdump* [34]. We recommend truncating the packets by setting the tcpdump's *snaplen* parameter (-s option) to 96 bytes as described in Section 5.3.5 to reduce the needed resources for the capturing process. Note that recording short time-period between in the range between 15 and 30 minutes is often sufficient (see Section 6.1.3). Next, the FlowMeter feature-extraction tool has to be applied to the recorded PCAP data. Finally, the generated CSV file containing the extracted flow-features is fed into the two inference stages which are implemented in a Python script.

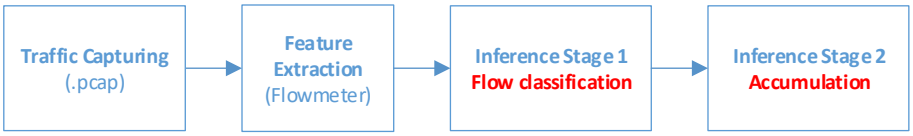


Figure 5.9: Architecture of the proposed IDS system

Requirements
In the following we summarize the requirements to deploy the proposed system:

- *Tcpdump*
- Java SE Runtime Environment 8 (to run FlowMeter)
- Python 2.7 / 3.6 with scikit-learn [65]

Output
The proposed IDS architecture can provide the following information to the defenders:

1. The IP addresses of the C&C listener servers owned by the red team
2. The IP addresses of the compromised hosts in the internal network
3. The ports used by the connections from the compromised hosts to the C&C servers

Having this information, the defender team can reconfigure the firewall to block the IP addresses belonging to the identified C&C servers. Furthermore the compromised machines can be put under quarantine while the situation is further investigated.

We emphasize that the proposed system is not designed to prevent exploits from happening but to detect hosts in the network that are already compromised. However, if the compromise is identified fast enough, the defender team can possibly prevent further lateral movement of the attackers through the network. Furthermore, by identifying and blocking the C&C servers, the resources available to the red team during the exercise can be dramatically decreased.

5.4 Unsupervised detection of malicious activities

Supervised-learning based detection systems are highly effective to detect those anomalies that are labelled in the training set. However, these systems cannot defend the network against new and unknown attacks. Moreover, labelling the malicious activities in network traffic is a challenging task and requires expert-knowledge.
Unsupervised models on the contrary are able to autonomously detect anomalies and to characterize traffic deviating from normal operation without relying on any prior information. As hackers continuously explore new ways to compromise systems and networks, the importance of developing such systems is rising.

While the final goal is to develop models that can perform without any prior knowledge on the data, in research it is crucial to have means to validate the function of the model. One way to do so, is by using a labelled dataset and using the ground truth to evaluate the prediction quality of the unsupervised model under investigation. As for the Locked Shields data we only obtained labels for a specific class of malicious activity (C&C sessions) and not for the remaining anomalies in the data, we decided to use the novel and fully labelled intrusion detection dataset CICIDS2017 [73] to investigate the unsupervised domain. After verifying the function of the unsupervised models we go back to the Locked Shields data to deploy them. Note that the CICIDS2017 dataset holds the same feature-set as the one we extracted from the Locked Shields data to train our supervised models that detect C&C sessions (see Section 5.3).

While there have been proposed different approaches to the unsupervised learning problem such as outlier detection, neural network based approaches or latent variable models, we choose to investigate unsupervised clustering algorithms in this work.
Namely we evaluated the performance of the classical K-means and the density-based DBSCAN algorithms, which we both described in Section 2.5.8.

5.4.1 The CICIDS2017 dataset

In Section 2.1.2 we provide already a short description of the CICIDS2017 dataset. Table 5.8 shows the different sample classes with their number of occurrence in the data.
We observe that while some attack types such as the DoS classes occur quite frequently, there are some classes such as Heartbleed, Infiltration and SQL Injection with very small sample counts.

5.4.2 Feature-selection

As the CICIDS2017 was created by the developers of the FlowMeter tool we used in the supervised detection task earlier in our investigation, the dataset comes with the exact same features, as listed in Appendix C.
To reduce the feature count of the CICIDS2017 dataset, we first run the feature elimination step as described in Section 5.3.2, where we remove features with zero or very little correlation to the labels. As unlike to the previous detection task where we had only two classes (normal, malicious), in this dataset we have a total of 15 classes. Before calculating the Pearson correlation and the Mutual Information scores between the feature vectors and the labels, we binarize the label vector by summarising all attack classes into one malicious class. By conducting this feature elimination process, we are able to remove 12 of the total 75 features listed in Table 5.9.

Class	Count	Fraction %
Benign	2271320	80.3189%
DDoS	128025	4.53724%
DoS GoldenEye	10293	0.36398%
DoS Hulk	230124	8.14769%
DoS Slowhttptest	5499	0.19445%
DoS slowloris	5796	0.20495%
FTP-Patator	7935	0.28059%
Heartbleed	11	0.00038%
Infiltration	36	0.00127%
PortScan	158804	5.62566%
SSH-Patator	5897	0.20853%
Web Attack - Brute Force	1507	0.05329%
Web Attack - Sql Injection	21	0.00074%
Web Attack - XSS	652	0.02305%
Bot	1956	0.07916%
	2827876	100%

Table 5.8: Class counts in the CICIDS2017 dataset

Low Correlation/MI
ECE Flag Count
RST Flag Count
CWE Flag Count
Fwd URG Flags
Bwd Avg Bytes/Bulk
Bwd Avg Packets/Bulk
Bwd URG Flags
Fwd Avg Bytes/Bulk
Fwd Avg Bulk Rate
Bwd PSH Flags
Fwd Avg Packets/Bulk
Bwd Avg Bulk Rate

Table 5.9: Features removed from the CICIDS2017 data in the feature-elimination step.

In preliminary experiments, we evaluated the recursive feature elimination algorithm as described in Section 5.3.2 to further reduce the feature count. However, reducing the number of features further did not improve the quality of the clusters obtained by both the K-means and DBSCAN algorithms. For this reason, we use the 63 features we obtain after executing the feature elimination step as described above to fit our final models.

However, for the final experiments using the Locked Shields LS17 and LS18 datasets we used the top-10 FlowMeter-features we identified in Section 5.3.2.

5.4.3 Model-selection

We start our analysis with the popular K-means clustering algorithm as described in Section 2.5.8. However, K-means comes with a number of limitations that might strongly affect performance, depending on the data being used:

1. The number of clusters K needs to be defined first. Finding the optimal K is not always a trivial task.

2. It can only find clusters with spherical shape.

3. It depends strongly on the initialization of the cluster-centers. Convergence to a global optimum is not guaranteed.

Despite these drawbacks, the K-means algorithm has shown to perform well on a variety of different tasks [41]. Even so, we select a second clustering algorithm DBSCAN for comparison. Being a density-based algorithm, unlike K-means, DBSCAN has the ability to detect clusters of arbitrary shapes. The main limitation of DBSCAN is that it cannot handle data containing clusters with strongly varying densities, due to its density based definition of the core points.

In our experiments on CICIDS2017, both algorithms achieved comparable performance (see Section 6.2.1).

5.4.4 Data preprocessing

Before fitting the clustering models, we applied standardization as a preprocessing step to the feature vectors as described in Section 2.5.4. As both algorithms, K-means and DBSCAN rely on euclidian distance measures, it is crucial to bring all features to the same scale.

5.4.5 Runtime issues

The CICIDS2017 dataset consists of more than 2 million samples. While the K-means algorithm has only linear complexity, we found that the DBSCAN algorithm which has quadratical complexity does not scale on the full dataset.

Subsampling
In order to evaluate the DBSCAN algorithm, we decided to generate smaller datasets by applying subsampling to the original data.
We applied two different subsampling strategies:

Uniform-subsampling We randomly select a subset of 10% of all samples from the full dataset, resulting in a smaller set of only 282787 samples.

Log-subsampling We reduce the sample count of each attack class in the data using a logarithmic function:

$$NrSamples_{new} = n \log(k \cdot NrSamples_{old}) \tag{5.2}$$

However, to keep the number of samples of the Benign class dominant, instead of using the above logarithmic downscaling we apply uniform subsampling selecting 0.1% of all Benign samples. Using this process with $n = 20$ and $k = 1000$ we obtain a dataset with only 5896 samples.

The drawback of the Uniform-subsampling process is that attack classes with very small sample counts will almost or completely vanish. The Log-subsampling method on the other hand reduces only the sample counts of the dominant classes, while not affecting the rare classes, as the logarithmic function can be approximated as a linear function for small arguments. In Table 5.10, we list the sample counts of the different classes for the two described subsampling methods. We observe that the Benign class is less dominant in the log-subsampled dataset and the variance of the different class sizes is significantly smaller than when using uniform subsampling. Note that while we used the attack labels in order to perform the Log-subsampling, no information about the classes is required for the Uniform-subsampling method. Thus in a purely unsupervised scenario where no ground truth is available, the Log-subsampling cannot be used. Even so, we found the results deduced from running our models on the Log-subsampled set to be of great interest to our investigation (see Section 6.2.1).

5.4.6 Performance measures

To evaluate the unsupervised clustering models, we make use of the ground truth in our dataset. First assign each cluster to the class which occurs most frequently in this cluster. We then define two measures which we call cluster **Purity** and **Detection-rate** to evaluate the performance of a clustering-model:

	Log		Uniform	
Class	**Count**	**Fraction**	**Count**	**Fraction**
Benign	2271	38.52%	227054	80.29153%
DDoS	374	6.34%	12740	4.50516%
DoS GoldenEye	323	5.48%	957	0.33842%
DoS Hulk	386	6.55%	23324	8.2479%
DoS Slowhttptest	311	5.27%	562	0.19874%
DoS slowloris	312	5.29%	561	0.19838%
FTP-Patator	318	5.39%	795	0.28113%
Heartbleed	11	0.19%	2	0.00071%
Infiltration	36	0.61%	4	0.00141%
PortScan	378	6.41%	15796	5.58583%
SSH-Patator	312	5.29%	590	0.20864%
Web Attack - Brute Force	285	4.83%	162	0.05729%
Web Attack - Sql Injection	21	0.36%	1	0.00035%
Web Attack - XSS	268	4.55%	60	0.02122%
Bot	290	4.92%	179	0.0633%
	5896	100%	282787	100%

Table 5.10: Class counts in the CICIDS2017 dataset after applying Log-subsampling and Uniform-subsampling

Purity We define the cluster Purity for a particular class as follows: We look at all clusters that have been assigned to the same class and then calculate the fraction between the number of the most-frequent class samples and the total number of samples in these clusters.

Detection-rate When looking at all clusters assigned to a particular class, the cluster detection-rate score calculates the fraction between the number of samples of this class that belong to these clusters and the total number of samples in the dataset of this class. In other words it measures what fraction of a particular class lies in the correct clusters and is thus detected.

5.4.7 Parameter tuning

In machine-learning choosing the optimal set of hyper-parameters for the learning algorithm is a crucial step to achieve best performance. In the following, we describe how we chose the parameters of our K-means and DBSCAN models.

5.4.7.1 Choosing the optimal K for K-means

A common difficulty that arises when using K-means is the selection of the number of clusters. The **Elbow-method** [43] is a common approach to find good K values.
The idea is to fit multiple K-means models on the data using different K values, while calculating a cost measure for each K. Often, up to some K value the cost drops dramatically until it reaches a plateau where cost decreases significantly less with increasing K than before. It is called Elbow-method because the resulting curve usually has an elbow like shape where the point for the optimal K value lies in the elbow.
We use the Sum of Squared Errors (SSE) [43] as cost function, which describes the sum of squared distances of all samples to their closest cluster center.

Figure 5.10 shows the resulting elbow curve, where the K value ranges from 2 to 30. We locate the elbow point somewhere between K=10 and K=15, which matches well the number of classes (15) in our detection task. However, we found in our experiments that the algorithm performed significantly better for higher K values (see Table 6.10).

5.4.7.2 Setting the DBSCAN parameters

As described in Section 2.5.8 there are two important parameters in DBSCAN that have to be defined before training; Epsilon and minSamples.

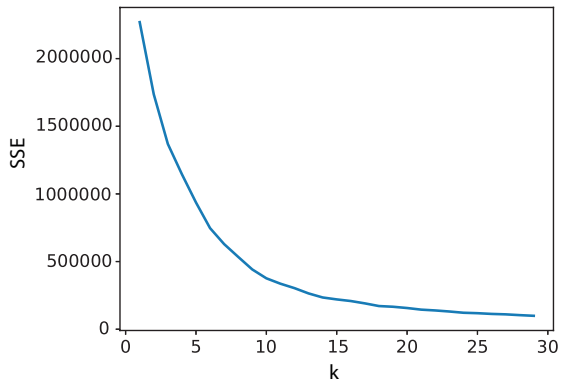


Figure 5.10: K-means elbow curve

Epsilon specifies the minimum distance between two points such as to be considered a part of a cluster while minSamples measures how many neighbours a point needs to have to be included into a cluster.

We found the configuration with Epsilon=3 and minSamples=3 to work best in our configuration. As we can observe in Table 6.12 in Chapter 6, when increasing these values the model starts to miss most of the classes in the dataset.

5.4.8 Problem with high number of clusters

In our experiments, we found that the results using K-means clustering improved significantly with increasing K value (see Table 6.10). However, if we assume a truly unsupervised scenario where we have no prior information about the malicious activities in the data, the higher the number of clusters the more difficult it gets for the analyst to interpret all the different clusters.

One reason we found for the better performance when using bigger K values is that the K-means algorithm strongly depends on the initialization of the cluster-centers. The main issue we observe is that for small K values, clusters for the less frequent classes in the dataset are hardly found. If we increase the number of cluster centers, we raise the probability of a cluster to hit one of the less prominent classes. However, this way we also introduce redundant cluster centers that will be assigned to the more frequent classes. For instance, when running K-means on the full CICIDS2017 dataset with K=100, 81 of the 100 classes are be assigned to the Benign class, which is clearly the most dominant class, accounting for 80% of the samples in the data.

Based on this observation, we introduce a simple algorithm designed to reduce the number of redundant clusters by merging cluster-centers that lie particularly close together. We first run K-means on the data using a high K value and then feed the resulting cluster-centers into a second stage where we run the cluster-reducing algorithm (see Figure 5.11). To decide which centers to merge, the algorithm uses a distance-threshold parameter. All cluster-centers whose pairwise distances lie below the configured distance-threshold will be merged by calculating the centroid of those centers. This centroid will replace the cluster-centers that have been merged in the new model.
The higher the distance-threshold parameter is set, the more the K value will be reduced in the new model. However, when setting the threshold too big, the algorithm will start to merge clusters belonging to different classes.

5.4.9 Unsupervised C&C detection in Locked Shields data

After performing the analysis of the K-means and DBSCAN algorithms on the CICIDS2017 data, we now evaluate the K-means model on the Locked Shields dataset. Even tough K-means and DBSCAN achieved comparable performance on CICIDS2017, we prefer the K-means algorithm due to its linear time complexity. K-means unlike DBSCAN thus scales to the Locked Shields data, even without subsampling the training data.

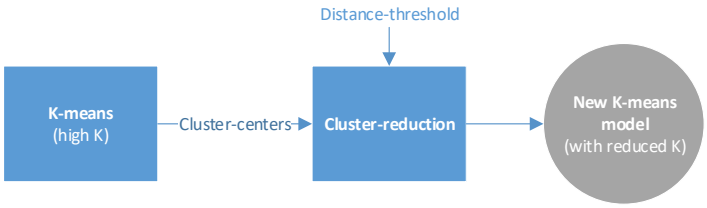


Figure 5.11: Two stage K-means approach to reduce the number of clusters.

We fit two K-means models on the LS17 and LS18 datasets respectively. Using a value of K=15, in both datasets we can assign exactly one one of the found clusters to the malicious class holding the C&C flows. While the C&C cluster found in the LS17 data has a fairly high purity value of 96.42%, the cluster in the LS18 has a purity of 75.36%. One possible explanation for the lower purity on the LS18 data is that due to the different background traffic distribution, the malicious samples might not as well separable in terms of euclidian distance as it is the case for the LS17 data.

Classification After identifying the cluster assigned to the malicious class, we evaluate if we can use the found cluster-centers in LS17 to detect malicious flows in LS18 and vice versa. To do so, we need to design a classifier model that bases its predictions on the found cluster-centers.

An obvious way to use the cluster-centers found by K-means to build a classifier is to classify those samples as malicious which lie closer to the malicious cluster-center than to all other centers. However, doing so we include also the cluster-centers of the normal class into the inference process, which in our case is not desirable, as the background traffic (normal class) in different Locked Shields exercises always varies. In the following we will refer to this classifier type as **K-Classifier**.

A more elegant way, as we only care about the malicious cluster, is to classify flows that are close to the found malicious cluster as malicious, while classifying samples farther away as normal. We base the predictions of the classifier on a distance-threshold. All the flows whose distances to the malicious center lie below the distance threshold we classify as malicious. We will refer to this type of classifier as **Threshold-Classifier**

To determine the optimal value of the distance-threshold, we evaluate the statistics of the pair-wise distances within the malicious clusters found in LS17 and LS18. Namely we calculate the min, max and mean distances of the malicious cluster-center to the samples belonging to the same cluster, as well as to the samples belonging to the other clusters. The aim is to set the threshold close to the mean distances within the malicious cluster. Table 5.11 lists these statistics for both the LS17 and the LS18 dataset. We observe that the mean euclidean distances to the malicious class samples is at 0.72 for the LS17 data and at 1.57 for the LS18 data. Based on this observation, we define the distance-threshold to be at a value of 1.0, which lies between the two mean values and which we found to perform well on both datasets.
We report the performance of the two proposed classification approaches in Table 6.15 in Chapter 6. The proposed Threshold-Classifier that we train in a fully unsupervised fashion, achieves 99% precision on the LS18 dataset and 83% precision when evaluated on the LS17 data, which is slightly below the performance scores of our supervised random forest model described in Section 5.3.

The only point, where we made use of the ground truth, was when we assigned the cluster found by K-means to the malicious class. In a real case scenario where no ground truth is available, this assignment of the found clusters can be conducted by a forensic analysis of the traffic in the different clusters. Our C&C cluster could be detected by identifying the periodicity of the flows going from the compromised machines to unknown external destination IP addresses.

By conducting this analysis, not only we showed that we can detect C&C sessions in a fully unsupervised fashion, but we have made also a proof of concept, that unsupervised approaches

	LS17 (K=15)	LS18 (K=15)
Distances to normal samples:		
min	0.07	0.49
max	1439.34	1438.6
mean	4.35	4.1
Distances to malicious samples:		
min	0.07	0.51
max	15.12	54.36
mean	0.72	1.57

Table 5.11: Statistics of the euclidian distance from the malicious cluster-centers to the samples of the same cluster (malicious) and to the samples of the other clusters (normal). The malicious center is obtained from the K-means models with K=15.

can be applied to the CICIDS2017 as well as the Locked Shields data in order to detect malicious activities in the traffic. While unsupervised approaches have been researched using the KDDCup99 and other datasets [46] [86], to the best of our knowledge, we are the first to investigate unsupervised intrusion detection using the novel FlowMeter feature-set.

Chapter 6

Evaluation

In this chapter, we evaluate the supervised and unsupervised models we described in Sections 5.3 and 5.4.

6.1 Supervised detection

6.1.1 Model selection

In Section 5.3.3, we described the model-selection strategy mainly consisting of two steps we follow to find the model-type that performs best on our data. In the following we report the results we measured during this model-selection process. All results reported in this Section will refer to the host-labelled versions of our datasets, as we discarded the session-labelling due to the reasons outlined in Section 5.3.3.
We used the scikit-learn 0.19.2 (*sklearn*) [65] machine-learning library with Python 3.6 to train the different models.

6.1.1.1 Step-1: Evaluation on LS17

We start with step-1 of the strategy, where we compare different supervised models on the ten LS17 training and validation sets generated as described in Section 2.5.2. Using the random forest based feature-selection algorithm as described in Section 5.3.2, we have already a list of all the features ordered by importance. So to select the best k features, we can just take the first k entries from this list (see Tables in C.4). However, we still have to find the optimal number of features for each supervised algorithm. To do so, we evaluate each of the supervised models on the training and validation sets we have generated, and repeat this for different feature counts to find the number of features where each model performs best. We tested the models for 58 (all), 40, 30, 20, 15 and 5 of the FlowMeter-features.

Tables 6.3 and 6.4 show the mean values and standard deviations of the precision and recall scores achieved by the different models over the ten different train and validation splits. Note that we train and evaluate the models for all of the ten dataset-splits in both directions, so in total we train and evaluate twenty models. To train the models in Table 6.3, we balanced the number of samples of the normal and malicious class in the training set, while the results in Table 6.4 result from training with the original, unbalanced data. We trained all models using a uniformly sampled subset (30%) of the LS17 data. This was particularly necessary for training the K-NN models, as due to their quadratic time complexity, they do not scale well on the full dataset.
We standardized the training and test sets in all cases except for training the Random Forest (RF) models, as they are invariant to standardization process.

We observe that the sweet spot for the number of features lies between 10 and 20 for all evaluated algorithms. While the scores of SVM, Logistic regression and Naive Bayes seem to strongly depend on the sizes of the different feature subsets, K-NN and random forest proof to be fairly invariant to the feature count.

Random forest and K-NN achieve the best results in this experiment, followed by Logistic Regression and SVM, which both achieved comparable scores. While the Naive Bayes model predicts above chance level, the results are not satisfactory. One reason might be that this model relies on the restrictive assumption that values associated with each class are distributed according to a Gaussian distribution. We prefer the SVM over the Logistic Regression model, as it performs better for lower feature counts, which reduces the required complexity of the model.

We further observe that while RF and K-NN perform better without applying class-balancing to the training set, the other models clearly benefit from this preprocessing step.

Using these results we can identify the optimal feature counts for all models, and if class-balancing improves performance. Table 6.1 lists the best configurations which we will also use as a starting point in the following step of the model-election strategy. Furthermore, we list in Table 6.2 the class names and the corresponding hyper-parameter configurations we used to train the models using the *scikit-learn* package.

Model	# features	Class-balancing
SVM	20	yes
Log. Regr.	15/20	yes
Naive Bayes	15	yes
K-NN	15	no
RF	10	no

Table 6.1: Configuration sweet-spots for all tested supervised models

Model	Class-name	Hyper-parameters
SVM	LinearSVC	loss='hinge'
Log. Regr.	LogisticRegression	max_iter=1000, penalty='l2', solver='sag'
Naive Bayes	GaussianNB	default
K-NN	KNeighborsClassifier	n_neighbors=5, weights='uniform'
RF	RandomForestClassifier	default

Table 6.2: Class names and the corresponding hyper-parameter configurations we used to train the models using the *scikit-learn* package.

#	SVM	Log. Regr.	Naive Bayes	K-NN	RF
58	0.69±0.22	0.71±0.19	0.37±0.23	0.64±0.23	0.69±0.23
	0.88±0.15	0.82±0.20	0.88±0.16	0.90±0.12	0.92±0.09
40	0.67±0.23	0.72±0.21	0.36±0.23	0.61±0.28	0.69±0.22
	0.81±0.21	0.82±0.21	0.90±0.15	0.86±0.18	0.88±0.16
30	0.68±0.20	0.71±0.21	0.36±0.23	0.63±0.27	0.69±0.21
	0.79±0.27	0.82±0.23	0.88±0.17	0.91±0.12	0.93±0.07
20	0.72±0.22	0.69±0.21	0.37±0.22	0.71±0.20	0.72±0.22
	0.79±0.26	0.82±0.23	0.90±0.13	0.85±0.22	0.91±0.09
15	0.69±0.21	0.69±0.21	0.51±0.29	0.74±0.19	0.72±0.22
	0.78±0.27	0.82±0.23	0.85±0.20	0.88±0.15	0.89±0.13
10	0.64±0.22	0.60±0.23	0.30±0.20	0.72±0.19	0.74±0.19
	0.78±0.28	0.77±0.26	0.79±0.28	0.94±0.08	0.93±0.0.08
5	0.27±0.23	0.26±0.18	0.17±0.18	0.60±0.23	0.62±0.23
	0.85±0.28	0.84±0.26	0.85±0.29	0.90±0.12	0.89±0.13

Table 6.3: LS17 evaluation of all supervised models using different feature subsets with class-balancing in the training set. The first row in each field of the table refers to the precision, while the second row refers to the recall metric (mean±std.)

#	SVM	Log. Regr.	Naive Bayes	K-NN	RF
58	0.52±0.30	0.09±0.11	0.17±0.12	0.84±0.21	0.81±0.26
	0.6±0.3	0.34±0.34	0.95±0.06	0.92±0.08	0.76±0.22
40	0.53±0.34	0.13±0.22	0.2±0.12	0.82±0.23	0.81±0.26
	0.76±0.28	0.52±0.43	0.91±0.15	0.92±0.08	0.78±0.23
30	0.63±0.32	0.13±0.22	0.21±0.12	0.82±0.23	0.84±0.23
	0.7±0.29	0.52±0.43	0.89±0.17	0.92±0.08	0.86±0.13
20	0.58±0.32	0.12±0.21	0.24±0.13	0.83±0.22	0.82±0.23
	0.78±0.23	0.52±0.43	0.84±0.24	0.9±0.08	0.81±0.22
15	0.63±0.32	0.13±0.22	0.43±0.27	0.85±0.21	0.85±0.22
	0.74±0.29	0.52±0.43	0.82±0.24	0.9±0.11	0.83±0.22
10	0.31±0.37	0.13±0.22	0.48±0.28	0.83±0.21	0.86±0.2
	0.31±0.35	0.55±0.45	0.82±0.24	0.87±0.13	0.88±0.13
5	0.17±0.28	0.0±0.0	0.18±0.14	0.71±0.29	0.8±0.23
	0.36±0.36	0.0±0.0	0.98±0.02	0.7±0.14	0.86±0.14

Table 6.4: LS17 evaluation of all supervised models using different feature subsets without class-balancing in the training set. The first row in each field of the table refers to the precision, while the second row refers to the recall metric (mean±std.)

6.1.1.2 Step-2: LS17 vs. LS18

After selecting the best performing models and finding their optimal configuration we proceed to step-2 of our model-selection strategy, where we train the models on the complete LS17 dataset and then evaluate performance on the full LS18 set and vice versa.

Notation In the following, we will refer to models trained on the full LS17 or LS18 datasets as **LS17-models** and **LS18-models** respectively. If we list performance scores for LS17-models, it implies that the evaluation was performed on the LS18 data and vice versa. We also provide the number of features used for training in brackets. If not specified differently, the number of features refers to the list of all features ordered by importance in Table C.4. For instance, **RF-LS17 (10F)** refers to a random forest model, trained on LS17 using the first 10 features from Table C.4 and evaluated on the full LS18 dataset.

In Table 6.5, we list the precision and recall scores for the best models and configurations identified in step-1 of our model-selection strategy. While we trained the RF and SVM models on the full datasets, we fitted the K-NN model on 30% uniformly subsampled training sets. We observe that the random forest model achieves high precision and recall values in both directions, while the performance of the K-NN and SVM models is significantly lower.

Model	Precision	Recall
RF-LS17 (10F)	0.98	0.98
RF-LS18 (10F)	0.98	0.84
RF-LS17 (20F)	0.95	0.98
RF-LS18 (20F)	0.99	0.92
KNN-LS17 (15F)	0.72	0.98
KNN-LS18 (15F)	0.97	0.48
SVM-LS17 (20F)	0.55	0.56
SVM-LS18 (20F)	0.73	0.44

Table 6.5: Precision and recall scores for the best models and configurations identified in step-1 of our model-selection strategy

6.1.1.3 Discarding the KDD-features

We have found that random forest models work particularly well on our classification task. So far we have reported solely results of models that we trained with the FlowMeter-features. In the

following, we repeat step-1 and step-2 of the model-selection strategy using a random forest model with the KDD feature-set.

Figure 6.1 shows the boxplots of the results after the evaluation on LS17 (step-1) using the top-20 FlowMeter versus the top-20 KDD features. We observe that the model relying on the FlowMeter-features performs significantly better than the KDD-model, which fails to achieve decent results in most of the 10 training and validation sets.

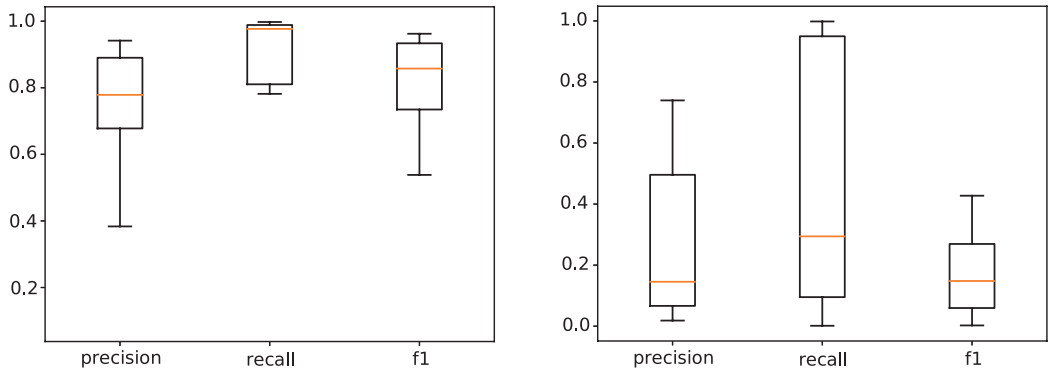


Figure 6.1: Comparison of the results of model-selection step-1 with the top-20 FlowMeter features (left) vs. the top-20 KDD-features (right) using a random forest model

Next, we run step-2 of the model-selection model comparing the same model configurations. Table 6.6 reports the precision and recall scores of this experiment. We observe that while the KDD LS17-model behaves very poorly, the LS18-model achieves a fairly high precision value of 0.93. However, recall is only at 0.05 meaning that the model only detects 5% of all the malicious flows. The FlowMeter models clearly outperform the KDD models, reaching high prediction and recall scores on both datasets.

Based on these findings, we conclude that the FlowMeter-features are more suitable our classification task, which is why we use these features rather than the popular KDD-features to develop our final IDS architecture.

Features	Model	Precision	Recall
KDD	**RF-LS17 (20F)**	0.37	0.03
FlowMeter	**RF-LS17 (20F)**	0.95	0.98
KDD	**RF-LS18 (20F)**	0.93	0.05
FlowMeter	**RF-LS18 (20F)**	0.99	0.92

Table 6.6: Comparison of results of model-selection step-2 with the top-20 FlowMeter features vs. the top-20 KDD-features using a random forest model

6.1.1.4 Selecting the best model

We found in step-2 of our model-selection process that the random forest algorithm performs best on our data. However, while the results from step-1 indicate that the optimal feature count to be at 10, Table 6.5 listing the results from step-2 suggests that the RF-LS18 with 20 features performs slightly better in terms of recall.
Even so, we prefer the model with only 10 features to the 20-features model for the following reasons:

1. In the results of step-1 (Table 6.4), the variance of the 10F model's recall is significantly lower than for the 20F model (0.13 vs. 0.22).
2. In step-2, the precision of the RF-LS17 (10F) model beats the precision of the 20F model by 3%. Note that in our task precision is more relevant than recall (see Section 5.3.1).
3. The computational complexity of the 10F model is significantly lower.

6.1.2 Attacking the model

In the model-selection process as described above, we found that a random forest classifier trained with the top-10 FlowMeter features works best in our classification task. In Section 5.3.4 we described the methodology we followed to analyse and improve the robustness of the model and its resilience to possible attacks.
In this Section we report the performance scores of the original model as well as the tuned models under attack.

First briefly recap how the tuned model configuration differs the original model:

- SYN/PSH Flag Cnt features replaced by Dst IntExt and Protocol features
- Tree count increased from 10 to 128
- Maximum tree depth constraint set to 10

We attack all the features of the random forest (10F) model we identified to be suitable attack vectors in Section 5.3.4. Then we penetrate each of the features separately, but launch also a 'COMBINED' attack which alters the feature values of multiple features at the same time (SYN Flag Cnt, Flow IAT Max, Flow Pkt/s, Flow IAT Mean, PSH Flag Cnt, Tot Fwd Pkts). Finally, we attack the features all together at the same time ('ALL ATTACKS'). Table 6.7 lists the precision and recall scores of the original and the tuned models using the top-10 and top-20 FlowMeter-features. In the following we summarize the most important findings that can be deduced from these results:

- The tuned model versions proof to be much more resilient to the attacks in general.
- The models with 20 features (20F) perform significantly better than the corresponding 10F models under all attacks.
- When launching the COMBINED attack, the original 10F models completely break down, while the tuned LS17-model still reports high precision.
- The tuned 20F model performs significantly better when launching all attacks together compared to the original 20F model .

Running the LS17 and LS18 models in parallel
From Table 6.7 we can deduce that two particularly sensitive attack vectors in the tuned 10F model are the features Tot Fwd Pkts and Init Fwd Win Byts. However, when attacking the Tot Fwd Pkts feature, the 10F LS17-model still performs well while the recall of the LS18-model is affected much stronger. When attacking Init Fwd Win Byts feature it is the other way around, as the LS18-model performs well while the LS17 is more affected. This leads us to the conclusion that by running the two models in parallel we achieve even more resilience to possible attacks. We provide an intuitive reasoning for this observation in the following:

We know that the distribution of the background traffic (normal class) in the LS17 dataset differs the background distribution in LS18 as the network architecture, the services and operating systems in use are not identical. This leads to slightly different node conditions in the decision trees of the two random forest ensembles.
The conclusion is that the attacker has to hit slightly different values for the LS17- and LS18-models to make them misclasify a sample as normal instead of malicious.

	ORIGINAL				TUNED			
Attack	**RF-LS17 (10F)**	**RF-LS18 (10F)**	**RF-LS17 (20F)**	**RF-LS18 (20F)**	**RF-LS17 (10F)**	**RF-LS18 (10F)**	**RF-LS17 (20F)**	**RF-LS18 (20F)**
(None)	0.98/0.98	0.98/0.84	0.95/0.97	0.99/0.92	0.98/0.95	0.99/0.85	0.99 0.97	0.98 0.84
SYN Flag Cnt	0.94/0.32	0.96/0.29	0.94/0.74	0.99/0.92	invariant	invariant	invariant	invariant
Fwd IAT Max	0.98/0.98	0.98/0.85	0.95/0.98	0.99/0.92	0.98/0.98	0.99/0.84	0.99 0.98	0.98 0.75
Flow Pkts/s	0.98/0.98	0.98/0.70	0.95/0.98	0.99/0.92	0.98/0.95	0.98/0.78	0.99 0.98	0.98 0.83
Flow IAT Mean	0.98/0.97	0.98/0.62	0.95/0.94	0.99/0.92	0.98/0.97	0.98/0.75	0.99 0.98	0.98 0.83
PSH Flag Cnt	0.97/0.62	0.97/0.47	0.94/0.85	0.99/0.91	invariant	invariant	invariant	invariant
Tot Fwd Pkts	0.98/0.93	0.98/0.57	0.95/0.96	0.99/0.92	**0.98/0.97**	**0.96/0.32**	0.99 0.98	0.98 0.76
COMBINED	0.17/0.00	0.00/0.00	0.93/0.67	0.94/0.21	**0.98/0.74**	0.00/0.00	0.99 0.99	0.97 0.58
Init Fwd Win Byts	0.97/0.60	0.96/0.31	0.93/0.67	0.97/0.44	**0.93/0.22**	**0.98/0.72**	0.99 0.91	0.97 0.45
Bwd Pkt Len Min	0.98/0.98	0.98/0.84	0.95/0.97	0.99/0.92	0.98/0.95	0.99/0.84	0.99 0.97	0.98 0.84
Active Mean	0.98/0.98	0.98/0.83	0.95/0.97	0.99/0.92	0.98/0.95	0.99/0.84	0.99 0.97	0.98 0.84
ALL ATTACKS	0.00/0.00	0.00/0.00	0.87/0.36	0.84/0.06	0.43/0.01	0.00/0.00	0.99 0.97	0.97 0.45

Table 6.7: Precision/Recall scores of the original models as well as the tuned models under attack.

6.1.3 Testing the IDS system

In the following, we evaluate the final IDS system using the two consequent inference stages as described in Section 5.3.6.

Instead of running the system on the full traffic captures of the entire exercise, we split the complete PCAP files into smaller files each holding the network traffic during a time period of 15 minutes to simulate a more realistic scenario for a possible deployment.
We then select a short time-period still at the beginning of the LS18 exercise to test how many IP addresses belonging to the red team's C&C servers and how many compromised machines in our network we can detect using our IDS system at this point in time. After running our system on a time period of only 30 minutes between 11:31am and 12:01pm on the first day of the exercise, 10 out of the 12 destination IP addresses at the top of the sorted list generated by the accumulation stage of the system are listed in the Cobalt Strike reports, thus belonging to the red team's infrastructure. We further observe 5 different source IP addresses belonging to the local network communicating with these servers. We conclude that these machines have been most likely compromised by the red team at this time.

This result proves the functionality of our proposed system in future Locked Shields exercises. Remember that we used the LS17 dataset as training data and then deployed the IDS system on the previously unseen LS18 data. Moreover, we achieved comparable results doing the experiment in the other direction, training the model on LS18 and running the IDS on the LS17 data. During the LS18 exercise, the blue team would have benefited greatly from the availability of this tool. They could have blocked already in the beginning of the exercise ten of the red team's servers and identified five infected hosts in the network. The fact, that the system works both for the LS17 and LS18, makes the solution promising for the deployment in future Locked Shields exercises.

6.1.4 Runtimes

In this section, we list the runtimes of the feature-extraction process using on the LS17 and LS18 datasets. Also, we list the training and inference times comparing the three supervised algorithms that performed best on our data; RF, SVM, K-NN. We conducted all experiments on a machine with 10 Intel Xeon E5-2699 cores and 16GB RAM.

Feature extraction

In Table 6.8, we report the runtimes of the FlowMeter tool using the LS17 and LS18 datasets. Note that these times refer to the extraction of the flow-statistic for the complete two days of the Locked Shields exercises. However, in Section 6.1.3 our model performed well on traffic captures from a time period of only 30 minutes. The extraction of the features for this shorter time period takes less than two minutes.

	Runtime	**# Flows**
LS17	2532s	16231754
LS18	5116s	28978112

Table 6.8: Feature-extraction runtimes using the FlowMeter tool for the LS17 and LS18 datasets

Training and inference

In Table 6.9, we list the training and inference times for the supervised algorithms we found to perform best in this classification task (see Section 6.1.1). For random forest, we list the runtimes both for the original configuration with only 10 features, as well as the tuned configuration with 20 features as described in Section 5.3.4. We trained the RF and SVM models on the full datasets, while K-NN was trained only on a uniformly subsampled LS17 set (30%).

We observe that the parameter tuning of RF increases training and inference time significantly. However, the inference runtime is still more than satisfactory for deployment in a real-case

scenario. The tuned RF model classifies flows in pcap recordings of the complete two days of the LS18 exercise in less then a minute.

	Training (LS17)	Inference (LS18)
original RF (10F)	94.23s	5.85s
tuned RF (20F)	1086.72s	48.51s
SVM (20F)	1191.55s	1.19s
K-NN (15F)	23578.58s	18000.76s

Table 6.9: Training and inference times of the selected supervised models. Note that K-NN was trained only a subset (30%) of the full LS17 set.

6.2 Unsupervised detection

In this Section, we evaluate the detection approaches based on the two unsupervised clustering algorithms K-means and DBSCAN such as described in Section 5.4. We start with the evaluation on the CICIDS2017 dataset followed by the results on the Locked Shields data.

6.2.1 CICIDS2017

6.2.1.1 Model selection

Table 6.10 lists the cluster purity and detection-rate measures on the full CICIDS2017 data using different K values. We observe that for small K, the model finds only clusters belonging to the most frequent classes in the data such as Benign or DoSHulk. However, with increasing K the model starts to identify more and more of the smaller clusters.

Class	k=15	k=30	k=100	k=300
Benign	0.88/0.99	0.93/0.94	0.94/0.97	0.96/0.97
DDoS	0.59/0.47	0.74/0.47	0.79/0.61	0.91/0.83
DoSGoldenEye	0.00/0.00	0.00/0.00	0.74/0.09	0.79/0.56
DoSHulk	0.84/0.61	0.86/0.61	0.97/0.60	0.88/0.75
DoSSlowhttptest	0.00/0.00	0.00/0.00	0.00/0.00	0.81/0.64
DoSslowloris	0.00/0.00	0.00/0.00	0.82/0.30	0.95/0.49
FTP-Patator	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
Heartbleed	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
Infiltration	0.00/0.00	0.00/0.00	0.83/0.14	0.83/0.14
PortScan	0.00/0.00	0.53/0.98	0.68/0.98	0.77/0.98
SSH-Patator	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
WebAttack-BruteForce	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
WebAttack-Sql Injection	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
WebAttack-XSS	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
Bot	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00

Table 6.10: Cluster Purity/Detection-rate of the K-means algorithm on the full CICIDS2017 dataset using different K values

Next we compare the performance of K-means using a K value of 15 on the Log-subsampled and the full dataset. The results are shown in Table 6.11. We observe that the algorithm performs much better on the Log-subsampled version of the dataset than on the full set. The reason we found for this result, is that the Benign class is less dominant in the Log-subsampled dataset and the variance of the sample counts of the different classes is significantly smaller than in the full dataset, which facilitates the clustering task significantly.

In Table 6.12 we show the performance of the DBSCAN algorithm on the Log-subsampled dataset in three different configurations. When increasing the Epsilon and the minSamples parameters, the model starts to miss most of the classes in the dataset. We identify Epsilon=3 and minSamples=3 to be the optimal parameter choice in this task.

	K=15	
	Log	**Full**
BENIGN	0.47/0.89	0.88/0.99
DDoS	0.45/0.48	0.59/0.47
DoSGoldenEye	0.93/0.11	0.00/0.00
DoSHulk	0.69/0.58	0.84/0.61
DoSSlowhttptest	0.80/0.73	0.00/0.00
DoSslowloris	0.73/0.48	0.00/0.00
FTP-Patator	0.42/0.47	0.00/0.00
Heartbleed	0.00/0.00	0.00/0.00
Infiltration	1.00/0.19	0.00/0.00
PortScan	0.00/0.00	0.00/0.00
SSH-Patator	0.00/0.00	0.00/0.00
WebAttack-BruteForce	0.00/0.00	0.00/0.00
WebAttack-Sql Injection	0.00/0.00	0.00/0.00
WebAttack-XSS	0.00/0.00	0.00/0.00
Bot	0.00/0.00	0.00/0.00

Table 6.11: Cluster Purity/Detection-rate of K-means (K=15) on the Log-subsampled and the full CICIDS2017 datasets

	Epsilon=10 minSamples=5	**Epsilon=5 minSamples=4**	**Epsilon=3 minSamples=3**
BENIGN	0.39/0.99	0.45/0.93	0.44/0.91
DDoS	0.00/0.00	0.90/0.09	0.94/0.13
DoSGoldenEye	0.00/0.00	1.00/0.05	1.00/0.09
DoSHulk	0.00/0.00	0.89/0.56	0.91/0.56
DoSSlowhttptest	1.00/0.02	0.96/0.82	0.97/0.86
DoSslowloris	0.67/0.01	0.95/0.51	0.98/0.51
FTP-Patator	0.00/0.00	0.50/0.46	0.55/0.46
Heartbleed	1.00/1.00	1.00/0.91	1.00/0.73
Infiltration	0.00/0.00	0.89/0.22	1.00/0.28
PortScan	0.00/0.00	0.00/0.00	0.00/0.00
SSH-Patator	0.00/0.00	0.00/0.00	0.00/0.00
WebAttack-BruteForce	0.00/0.00	0.74/0.05	1.00/0.05
WebAttack-Sql Injection	0.00/0.00	0.00/0.00	0.00/0.00
WebAttack-XSS	0.00/0.00	0.00/0.00	1.00/0.02
Bot	0.00/0.00	0.00/0.00	0.00/0.00
Nr. clusters	6	32	48

Table 6.12: DBSCAN evaluated on the Log-subsampled dataset trained with different parameters. Scores refer to cluster Purity/Detection-rate.

Next, we compare the results of DBSCAN fitted on the Log-subsampled dataset with the performance achieved on the Uniform-subsampled data. Remember that we cannot evaluate this algorithm on the full dataset due to its quadratic computation complexity. So the closest we can get to the evaluation on the complete dataset is to fit the model on a uniformly subsampled version.

Table 6.13 lists the performace scores of DBSCAN for the Log-subsampled and the Uniform-subsampled datasets. We observe that the number of clusters for the uniform subsampled set is significantly higher (222 vs. 48). In order to make a fair comparison to the K-means algorithm, we fit a new model on the Uniform-subsampled data while setting the K value to 222 to match the number of clusters with the DBSCAN model. We find the K-means algorithm to perform slightly better on this data. We also prefer the K-means model to the DBSCAN algorithm due to its linear time complexity.

	DBSCAN		K-means
	Log	Uniform	Uniform
BENIGN	0.44/0.91	0.85/1.00	0.95/0.98
DDoS	0.94/0.13	0.87/0.10	0.92/0.82
DoSGoldenEye	1.00/0.09	1.00/0.03	0.87/0.52
DoSHulk	0.91/0.56	0.93/0.60	0.93/0.71
DoSSlowhttptest	0.97/0.86	0.91/0.32	0.72/0.61
DoSslowloris	0.98/0.51	0.98/0.53	0.91/0.48
FTP-Patator	0.55/0.46	0.00/0.00	0.00/0.00
Heartbleed	1.00/0.73	0.00/0.00	0.00/0.00
Infiltration	1.00/0.28	0.00/0.00	0.00/0.00
PortScan	0.00/0.00	0.93/0.00	0.77/0.98
SSH-Patator	0.00/0.00	0.00/0.00	0.00/0.00
WebAttack-BruteForce	1.00/0.05	1.00/0.07	0.00/0.00
WebAttack-Sql Injection	0.00/0.00	0.00/0.00	0.00/0.00
WebAttack-XSS	1.00/0.02	0.00/0.00	0.00/0.00
Bot	0.00/0.00	0.00/0.00	0.00/0.00
Nr. Clusters	48	222	222

Table 6.13: Cluster Purity/detection-rate scores of DBSCAN for the Log-subsampled and the Uniform-subsampled datasets compared with K-means.

6.2.1.2 Reducing the number of clusters

In the last Section we found that K-means acts slightly better on the CICIDS2017 than the DBSCAN algorithm.
However, we observed that the performance is only satisfactory for big K values ($K > 100$), which leads to a number of redundant cluster-centers particularly assigned to the Benign class. In the following, we evaluate the cluster-reduction algorithm as proposed in Section 5.4.8. First, we train a K-means model with K=100. Setting the distance-threshold parameter of our algorithm to 10, the number of clusters are reduced to 47. Table 6.14 shows the performance scores using two K-means models with K=50 and K=100, and the metrics of the 47 cluster model generated using the two stage approach. Comparing the K=50 to the reduced model, we observe that while the K=50 model finds no clusters for the DoSGoldenEye and DoSslowloris classes, the reduced model with 47 clusters preserves them as found by the K=100 model. Also, cluster purity of the DDoS class is 10% higher in the reduced model than in the K=50 model. In the reduced model, the PortScan cluster disappears which is found by the K=50 model. However, note that the purity of the PortScan clusters in the K=50 model is only at 59% which is not satisfactory for deploying the system in a real case scenario.

6.2.2 Locked Shields

In the following, we evaluate the detection rates of the two classifier types that rely on the malicious cluster-centers as described in Section 5.4.9. Table 6.15 shows the precision and recall prediction scores on the LS18 data of the classifiers obtained from the clusters found in the LS17 dataset and vice versa. We observe that the Threshold-Classifier reaches higher precision scores both on the LS17 and LS18 data.

Remember that our supervised random forest based model achieved 98% precision on both the LS17 and LS18 datasets. The unsupervised model we propose comes with 94% precision on the LS17 data very close to this prediction quality, while performing slightly worse on the LS18 data (83%). The performance can possibly be further increased by deploying more sophisticated unsupervised models than K-means.

	K=50	K=100	Reduction from K=100 Threshold=10
BENIGN	0.94/0.95	0.94/0.97	0.88/1.00
DDoS	0.67/0.56	0.79/0.61	0.77/0.55
DoSGoldenEye	0.00/0.00	0.74/0.09	0.72/0.09
DoSHulk	0.93/0.60	0.97/0.60	0.92/0.60
DoSSlowhttptest	0.00/0.00	0.00/0.00	0.00/0.00
DoSslowloris	0.00/0.00	0.82/0.30	0.79/0.30
FTP-Patator	0.00/0.00	0.00/0.00	0.00/0.00
Heartbleed	0.00/0.00	0.00/0.00	0.00/0.00
Infiltration	0.83/0.14	0.83/0.14	0.83/0.14
PortScan	0.59/0.99	0.68/0.98	0.00/0.00
SSH-Patator	0.00/0.00	0.00/0.00	0.00/0.00
WebAttack-BruteForce	0.00/0.00	0.00/0.00	0.00/0.00
WebAttack-Sql Injection	0.00/0.00	0.00/0.00	0.00/0.00
WebAttack-XSS	0.00/0.00	0.00/0.00	0.00/0.00
Bot	0.00/0.00	0.00/0.00	0.00/0.00
Nr. Clusters	50	100	47

Table 6.14: Cluster Purity/Detection-rate scores using two K-means models with K=50 and K=100, compared with a model where we reduced the clusters from K=100 to only 47 clusters using our cluster-reduction algorithm with a distance-threshold of 10.

	K-Classifier	Threshold-Classifier
LS17 Data	0.96/0.94	0.99/0.93
LS18 Data	0.74/0.56	0.83/0.56

Table 6.15: Precision and recall prediction scores on the LS18 data of the classifiers obtained from the clusters found in the LS17 dataset and vice versa.

Chapter 7

Conclusion and outlook

In this chapter, we summarize the main contributions made in this thesis. Finally, we provide an outlook containing possible future research directions that could be investigated to further improve the results of this thesis.

7.1 Conclusion

This thesis can be categorized mainly into three different parts which we describe in the following, while outlining the main contributions made in each part.

In the first part we investigate methods for performing effective analysis of network traffic. We build up an extensive database using Elasticsearch, containing different representations (packets, flows, alerts) of the network data from two past Locked Shields cyber-defense exercises. This set-up enables network traffic analysis on a large scale while enabling pivoting between the different levels of abstraction of the data. Forensic analysts could benefit greatly from such a system during their investigation.

In the second part of the thesis we develop a supervised model that is able to predict flows between compromised machines and C&C-servers with high precision. We further propose several means to increase the robustness of the model and show its resilience to a number of simulated attacks and packet loss. Finally, we present a scheme based on two inference stages in order to effectively deploy the model as a intrusion detection system during a Locked Shields exercise. In a demonstration we show that by running the system on a time-window of only 30 minutes during the beginning of Locked Shields 2018, we are able to identify a number of compromised machines and IP addresses of the C&C servers owned by the attacker team.

In the third part we analyse the recent CICIDS2017 dataset using unsupervised clustering algorithms. We successfully identify clusters for various different attack-types in the data while pointing out some the challenges as well as the limitations of the applied algorithms. To the best of our knowledge we are the first to evaluate unsupervised detection techniques on this novel dataset. Finally, we evaluate unsupervised clustering on the Locked Shields datasets. We are able to identify clusters with high purity for the C&C flows we previously classified using a supervised approach. We show how to precisely detect these sessions in a fully unsupervised fashion without relying on any prior information on the data.

7.2 Future Work

In the following, we provide an outlook containing possible future research directions that could be investigated in order to further improve the results of this thesis, but also to explore new closely related topics.

Implement sequence number checking

The current version of the FlowMeter tool is based on a very naive logic to track TCP

connection states and particularly for terminating TCP connections. By implementing sequence number checking in the FlowMeter tool, the packet injection attacks described in Section 5.3.4 could be prevented, making it much more challenging for a possible threat actor to attack the model.

Real-time deployment
The proposed IDS approach to detect the malicious C&C sessions works in a offline fashion and requires the processing of PCAP files holding recorded traffic from the network. An online solution that extracts the required features directly on the wire, while generating alerts in real-time would be beneficial. While FlowMeter offers a real-time feature-extraction mode, where the tool listens to the traffic on the network interface, we doubt that the tool could handle the load during the Locked Shields exercise. In future work, performance measurements could be conducted to find the maximal throughput the tool can handle. If our assumption is confirmed, a more efficient tool could be implemented, calculating only the features that our model really needs. Alternatively, the possibility to extract a new set of less costly features using flow export options such as *NetFlow*, *IPFIX* and *sFlow*) might be investigated.

Interpretation of clusters
In this work we successfully identified clusters belonging to a number of different attack classes. However, to validate the function of these models, we relied on prior information about the attacks. It would be interesting to explore methods to identify the type of flows belonging to a particular cluster without relying on a given ground truth. To achieve this, the network data analysis set-up using Elasticsearch could be extended by implementing mechanisms for pivoting between flows belonging to the clusters identified by the unsupervised algorithm and the data analysis framework, in order to interpret the information contained in the different clusters.

Unsupervised detection In this thesis, we investigated two standard unsupervised clustering algorithms; K-means and DBSCAN. One of the main issues we found the clustering approach is that most standard algorithms do not scale to big datasets. As in networking applications, it is common to work with big amounts of data, a future task might involve to identify or develop algorithms that converge also on big data. Moreover, it would be interesting to evaluate the performance of other unsupervised techniques such outlier detection, neural network based approaches or latent variable models on the CICIDS2017 and Locked Shields datasets.

Appendix A

Glossary

Flow/Connection A series of packets exchanged between two hosts, identifiable by the 5-tuple (source address, source port, destination address, destination port, transport protocol)

C&C session Refers to opened sessions between a compromised host and a C&C server owned by the threat actor.

Cobalt Strike Penetration testing software used by the red team during the Locked Shields exercise.

Locked Shields One of the largest and most complex international cyber-defence exercises in the world.

PCAP (Packet CAPture) Refers to network traffic captures stored in PCAP fileformat.

FlowMeter Tool that we use to extract flow statistics from network traffic captures.

Session-labelling Strategy for labelling Cobalt Strike's C&C sessions in the Locked Shields by matching the source, destination IPs and the listed timewindows of the sessions.

Host-labelling Strategy for labelling Cobalt Strike's C&C sessions in the Locked Shields by matching either the source or destination IP only with IP-addresses affiliated with the red team.

Acronyms

ANN Artificial Neural Networks

C&C Command and Control

CS Cobalt Strike

CSV Comma Separated Value

DPI Deep Packet Inspection

GUI Graphical User Interface

HIDS Host-based Intrusion Detection System

IDS Intrusion detection System

K-NN K-Nearest Neighbours

MI Mutual Information

NIDS Network-based Intrusion Detection System

SSE Sum of Squared Errors

SVM Support Vector Machine

VLAN Virtual Local Area Network

Appendix B

Elasticsearch mappings

pcap mapping

```
{
  "template": "packets-*",
  "mappings": {
    "pcap_file": {
      "_source": { "enabled": true },
      "dynamic": "false",
      "properties": {
        "timestamp": { "type": "date" },
        "protocol": { "type": "keyword" },
        "src_ip": { "type": "ip" },
        "dst_ip": { "type": "ip" },
        "src_port": { "type": "integer" },
        "dst_port": { "type": "integer" },
        "layers": {
          "properties": {
            "eth": {
              "properties": {
                "eth_eth_src": { "type": "keyword" },
                "eth_eth_dst": { "type": "keyword" }
              }
            },
            "ssl": {
              "properties": {
                "ssl_handshake_ssl_handshake_version": { "type": "keyword" },
                "ssl_record_ssl_record_length": { "type": "integer" }
              }
            },
            "frame": {
              "properties": {
                "frame_frame_len": { "type": "long" },
                "frame_frame_protocols": { "type": "keyword" }
              }
            },
            "ip": {
              "properties": {
                "ip_ip_src": { "type": "ip" },
                "ip_ip_dst": { "type": "ip" },
                "ip_ip_checksum_status": { "type": "byte" },
                "ip_ip_ttl": { "type": "long" }
              }
            },
            "ipv6": {
              "properties": {
                "ipv6_ipv6_plen": { "type": "integer" },
                "ipv6_ipv6_dst": { "type": "ip" },
                "ipv6_ipv6_flow": { "type": "keyword" },
                "ipv6_ipv6_src": { "type": "ip"}
              }
            },
            "tcp": {
              "properties": {
                "tcp_flags_tcp_flags_cwr": { "type": "byte" },
                "tcp_flags_tcp_flags_ack": { "type": "byte" },
                "tcp_flags_tcp_flags_syn": { "type": "byte" },
                "tcp_flags_tcp_flags_res": { "type": "byte" },
                "tcp_flags_tcp_flags_ns": { "type": "byte" },
                "tcp_tcp_seq": { "type": "long" },
                "tcp_tcp_stream": { "type": "integer" },
                "tcp_tcp_ack": { "type": "long" },
                "tcp_tcp_checksum_status": { "type": "byte" },
                "tcp_tcp_srcport": { "type": "integer" },
                "tcp_tcp_dstport": { "type": "integer" }
              }
```

```
            },
            "udp": {
              "properties": {
                "udp_udp_dstport": { "type": "integer" },
                "udp_udp_srcport": { "type": "integer" },
                "udp_udp_checksum_status": { "type": "byte" },
                "udp_udp_stream": { "type": "integer" }
              }
            },
            "http": {
              "properties": {
                "http_http_request_full_uri": { "type": "keyword" },
                "http_http_host": { "type": "keyword" },
                "http_response_code": { "type": "integer" },
                "http_response_code_desc": { "type": "keyword" },
                "http_http_request": { "type": "byte" },
                "http_http_response": { "type": "byte" }
              }
            },
            "dns": {
              "properties": {
                "dns_flags_dns_flags_opcode": { "type": "byte" },
                "dns_flags_dns_flags_checkdisable": { "type": "byte" },
                "dns_flags_dns_flags_recavail": { "type": "byte" },
                "text_dns_resp_name": { "type": "keyword" },
                "text_dns_resp_class": { "type": "keyword" },
                "text_dns_resp_type": { "type": "short" },
                "text_dns_qry_type": { "type": "short" }
              }
            }
          }
        }
      }
    }
  }
}
```

Bro mapping

```
{
  "template": "bro-*",
  "settings" : {
      "index" : {
          "number_of_shards" : 1,
          "number_of_replicas" : 0
      }
  },
  "mappings": {
    "bro": {
      "_source": { "enabled": true },
      "dynamic": "true",
      "properties": {
        "timestamp": { "type": "date" },
        "duration": { "type": "float" },
        "src_ip": { "type": "ip" },
        "dst_ip": { "type": "ip" },
        "src_port": { "type": "integer" },
        "dst_port": { "type": "integer" },
        "orig_pkts": { "type": "long" },
        "resp_pkts": { "type": "long" },
        "orig_ip_bytes": { "type": "long" },
        "resp_ip_bytes": { "type": "long" }
      }
    }
  }
}
```

Snort mapping

```
{
  "template": "snort-*",
  "settings" : {
      "index" : {
          "number_of_shards" : 1,
          "number_of_replicas" : 0
      }
  },
  "mappings": {
    "snort": {
      "_source": { "enabled": true },
      "dynamic": "true",
      "properties": {
        "timestamp": { "type": "date" },
        "src_ip": { "type": "ip" },
        "dst_ip": { "type": "ip" },
```

```
        "ethsrc": { "type": "keyword" },
        "ethdst": { "type": "keyword" },
        "src_port": { "type": "integer" },
        "dst_port": { "type": "integer" },
        "malicious": { "type": "keyword" }
      }
    }
  }
}
```

Appendix C

Network traffic features

C.1 KDDCup99 features

Nr.	Feature	Type	Description
1	duration	Integer	length (number of seconds) of the connection
2	protocol_type	Nominal	type of the protocol, e.g. tcp, udp, etc.
3	service	Nominal	network service on the destination, e.g., http, telnet, etc.
4	flag	Nominal	normal or error status of the connection (bro conn state)
5	src_bytes	Integer	number of data bytes from source to destination
6	dst_bytes	Integer	number of data bytes from destination to source
7	land	Binary	1 if src and dst IP addresses and port numbers are equal, else 0
8	wrong_fragment	Integer	number of “wrong” fragments
9	urgent	Integer	number of urgent packets (tcp urgent flag set)
10	Hot	Integer	number of “hot” indicators (e.g. entering directory, executing programs)
11	num_failed_logins	Integer	number of failed login attempts
12	logged_in	Binary	1 if successfully logged in; 0 otherwise
13	num_compromised	Integer	number of “compromised” conditions
14	root_shell	Binary	1 if root shell is obtained; 0 otherwise
15	su_attempted	Binary	1 if “su root” command attempted; 0 otherwise
16	num_root	Integer	number of “root” accesses
17	num_file_creations	Integer	number of file creation operations
18	num_shells	Integer	number of shell prompts
19	num_access_files	Integer	number of operations on access control files
20	num_outbounds_cmds	Integer	number of outbound commands in an ftp session
21	is_hot_login	Binary	1 if the login belongs to the “hot” list; 0 otherwise
22	is_guest_login	Binary	1 if the login is a “guest”login; 0 otherwise
23	count	Integer	#flows to the same source as the current flow (past 2s)
24	srv_count	Integer	#flows to the same service as the current flow (past 2s)
25	serror_rate	Float	#flows that have “SYN” errors
26	srv_serror_rate	Float	#flows that have “SYN” errors
27	rerror_rate	Float	#flows that have “REJ” errors
28	srv_rerror_rate	Float	#flows that have “REJ” errors
29	same_srv_rate	Float	#flows to the same service
30	diff_srv_rate	Float	#flows to different services
31	srv_diff_host_rate	Float	#flows to different hosts
32	dst_host_count	Integer	#flows to the same destination as the current flow (past 2s)
33	dst_host_srv_count	Integer	#flows to the same service as the current flow (past 2s)
34	dst_host_same_srv_rate	Float	#flows to the same service
35	dst_host_diff_srv_rate	Float	#flows to the different service
36	dst_host_same_src_port_rate	Float	#flows with source ports
37	dst_host_srv_diff_host_rate	Float	#flows to different hosts
38	dst_host_serror_rate	Float	#flows that have “SYN” errors
39	dst_host_srv_serror_rate	Float	#flows that have “SYN” errors
40	dst_host_rerror_rate	Float	#flows that have “REJ” errors
41	dst_host_srv_rerror_rate	Float	#flows that have “REJ” errors

Table C.1: Descriptions of the 41 features used in the KDDCup99 dataset [30]

C.2 FlowMeter features

Nr.	Feature	Type	Description
1	Flow Duration	Float	Duration of the flow in Microsecond
2	Tot Fwd Pkts	Integer	Total packets in the forward direction
3	Tot Bwd Pkts	Integer	Total packets in the backward direction
4	TotLen Fwd Pkts	Integer	Total size of packet in forward direction
5	TotLen Bwd Pkts	Integer	Total size of packet in backward direction
6	Fwd Pkt Len Min	Integer	Minimum size of packet in forward direction
7	Bwd Pkt Len Min	Integer	Minimum size of packet in backward direction
8	Fwd Pkt Len Max	Integer	Maximum size of packet in forward direction
9	Bwd Pkt Len Max	Integer	Maximum size of packet in backward direction
10	Fwd Pkt Len Mean	Float	Mean size of packet in forward direction
11	Bwd Pkt Len Mean	Float	Mean size of packet in backward direction
12	Fwd Pkt Len Std	Float	Standard deviation size of packet in forward direction
13	Bwd Pkt Len Std	Float	Standard deviation size of packet in backward direction
14	Fwd IAT Tot	Float	Total time between two packets sent in the forward direction
15	Bwd IAT Tot	Float	Total time between two packets sent in the backward direction
16	Fwd IAT Min	Float	Minimum time between two packets sent in the forward direction
17	Bwd IAT Min	Float	Minimum time between two packets sent in the backward direction
18	Fwd IAT Max	Float	Maximum time between two packets sent in the forward direction
19	Bwd IAT Max	Float	Maximum time between two packets sent in the backward direction
20	Fwd IAT Mean	Float	Mean time between two packets sent in the forward direction
21	Bwd IAT Mean	Float	Mean time between two packets sent in the backward direction
22	Fwd IAT Std	Float	Standard deviation time between two packets sent in the forward direction
23	Bwd IAT Std	Float	Standard deviation time between two packets sent in the backward direction
24	Fwd PSH Flags	Integer	#times the PSH flag was set in packets in the fwd direction (0 for UDP)
25	Bwd PSH Flags	Integer	#times the PSH flag was set in packets in the bwd direction (0 for UDP)
26	Fwd URG Flags	Integer	#times the URG flag was set in packets in the fwd direction (0 for UDP)
27	Bwd URG Flags	Integer	#times the URG flag was set in packets in the bwd direction (0 for UDP)
28	Fwd Header Len	Integer	Total bytes used for headers in the forward direction
29	Bwd Header Len	Integer	Total bytes used for headers in the forward direction
30	Fwd Pkts/s	Float	Number of forward packets per second
31	Bwd Pkts/s	Float	Number of backward packets per second
32	Flow Pkts/s	Float	Number of flow packets per second
33	Flow Byts/s	Float	Number of flow bytes per second
34	Pkt Len Min	Integer	Minimum packet length of a flow
35	Pkt Len Max	Integer	Maximum packet length of a flow
36	Pkt Len Mean	Float	Mean packet length of a flow
37	Pkt Len Std	Float	Standard deviation length of a flow
38	Flow IAT Min	Float	Minimum inter-arrival time of packet
39	Flow IAT Max	Float	Maximum inter-arrival time of packet
40	Flow IAT Mean	Float	Mean inter-arrival time of packet
41	Flow IAT Std	Float	Standard deviation inter-arrival time of packet
42	FIN Flag Cnt	Integer	Number of packets with FIN
43	SYN Flag Cnt	Integer	Number of packets with SYN
44	RST Flag Cnt	Integer	Number of packets with RST
45	PSH Flag Cnt	Integer	Number of packets with PUSH
46	ACK Flag Cnt	Integer	Number of packets with ACK
47	URG Flag Cnt	Integer	Number of packets with URG
48	CWR Flag Count	Integer	Number of packets with CWE
49	ECE Flag Cnt	Integer	Number of packets with ECE
50	Down/Up Ratio	Float	Download and upload ratio
51	Pkt Size Avg	Float	Average size of packet
52	Fwd Seg Size Avg	Float	Average size observed in the forward direction
53	Fwd Byts/b Avg	Float	Average number of bytes bulk rate in the forward direction
54	Fwd Pkts/b Avg	Float	Average number of packets bulk rate in the forward direction
55	Fwd Blk Rate Avg	Float	Average number of bulk rate in the forward direction
56	Bwd Seg Size Avg	Float	Average size observed in the backward direction
57	Bwd Byts/b Avg	Float	Average number of bytes bulk rate in the backward direction

58	Bwd Pkts/b Avg	Float	Average number of packets bulk rate in the backward direction
59	Bwd Blk Rate Avg	Float	Average number of bulk rate in the backward direction
60	Subflow Fwd Pkts	Float	The average number of packets in a sub flow in the forward direction
61	Subflow Fwd Byts	Float	The average number of bytes in a sub flow in the forward direction
62	Subflow Bwd Pkts	Float	The average number of packets in a sub flow in the backward direction
63	Subflow Bwd Byts	Float	The average number of bytes in a sub flow in the backward direction
64	Active Min	Float	Minimum time a flow was active before becoming idle
65	Active Mean	Float	Mean time a flow was active before becoming idle
66	Active Max	Float	Maximum time a flow was active before becoming idle
67	Active Std	Float	Standard deviation time a flow was active before becoming idle
68	Idle Min	Float	Minimum time a flow was idle before becoming active
69	Idle Mean	Float	Mean time a flow was idle before becoming active
70	Idle Max	Float	Maximum time a flow was idle before becoming active
71	Idle Std	Float	Standard deviation time a flow was idle before becoming active
72	Init Fwd Win Byts	Float	The total number of bytes sent in initial window in the forward direction
73	Init Bwd Win Byts	Float	The total number of bytes sent in initial window in the backward direction
74	Fwd Act Data Pkts	Float	Count of packets with at least 1 byte of TCP data payload in the forward direction
75	Fwd Seg Size Min	Float	Minimum segment size observed in the forward direction

Table C.2: Descriptions of the 75 features as generated by the FlowMeter tool [33]

After a thorough analysis of the provided feature descriptions and the source code of the FlowMeter tool, we came to the conclusion that some of the features are duplicated using different names. This concerns in particular the 'subflow' features 60-63. Unfortunately the authors of the tool don't define the term subflow in their documentation, so we could not add the missing implementation for these features. For instance we found that *Subflow Fwd Pkts* just calculates the total number of packets in the forward direction, so its equal to the *Tot Fwd Pkts* feature. Table C.3 lists all duplicated features we found:

Name 1	Name 2
Tot Fwd Pkts	Subflow Fwd Pkts
Tot Bwd Pkts	Subflow Bwd Pkts
TotLen Fwd Pkts	Subflow Fwd Byts
TotLen Bwd Pkts	Subflow Bwd Byts

Table C.3: Duplicated FlowMeter features

C.3 Features ordered by importance

Rank	FlowMeter-Features	KDD-Features
1	Tot Fwd Pkts	srv_diff_host_rate
2	Flow IAT Mean	dst_bytes
3	Fwd IAT Max	service
4	PSH Flag Cnt	flag
5	Flow Pkts/s	srv_count
6	Bwd Pkt Len Min	src_bytes
7	FIN Flag Cnt	count
8	Init Fwd Win Byts	same_srv_rate
9	SYN Flag Cnt	serror_rate
10	Active Mean	srv_serror_rate
11	Bwd IAT Mean	diff_srv_rate
12	Bwd Pkt Len Std	dst_host_count
13	Fwd Seg Size Min	dst_host_srv_count
14	Fwd Pkt Len Std	dst_host_srv_diff_host_rate
15	Tot Bwd Pkts	dst_host_srv_serror_rate
16	Bwd Header Len	dst_host_serror_rate
17	Tot Fwd Byts	dst_host_same_srv_rate

18	Tot Bwd Pkts	dst_host_diff_srv_rate
19	Fwd IAT Tot	protocol_type
20	Flow IAT Max	dst_host_same_src_port_rate
21	Pkt Len Min	rerror_rate
22	Pkt Len Mean	
23	Idle Mean	
24	Fwd IAT Mean	
25	TotLen Bwd Pkts	
26	Flow Duration	
27	Fwd IAT Std	
28	Down/Up Ratio	
29	Fwd Pkt Len Mean	
30	Flow IAT Std	
31	Pkt Len Max	
32	Tot Fwd Pkts	
33	ACK Flag Cnt	
34	Bwd Seg Size Avg	
35	Fwd Pkt Len Min	
36	Bwd IAT Max	
37	Bwd IAT Std	
38	Active Max	
39	Bwd Pkt Len Max	
40	Pkt Size Avg	
41	Active Min	
42	Fwd Header Len	
43	Bwd Pkts/s	
44	Pkt Len Var	
45	Subflow Bwd Byts	
46	Flow Byts/s	
47	RST Flag Cnt	
48	Fwd Seg Size Avg	
49	Fwd Pkt Len Max	
50	TotLen Fwd Pkts	
51	Idle Min	
52	Init Bwd Win Byts	
53	Bwd Pkt Len Mean	
54	Bwd IAT Tot	
55	Fwd Act Data Pkts	
56	Pkt Len Std	
57	ECE Flag Cnt	
58	Idle Max	

Table C.4: Features ordered by importance. List is obtained by following the feature selection strategy described in Section 5.3.2. Note that the constant/zero features that have been eliminated in the first step of the feature selection scheme are not listed.

Appendix D

Thesis proposal

Analyzing and understanding network activity can be extremely challenging in large and heterogeneous networks. In case of security incidents, response teams often invest an significant part of their resources to understand the compromised network architecture and to identify suspicious activity pattern for subsequent investigations. These tasks can be improved in order to allow response teams to focus on the incident resolution instead of consuming resources on legitimate activity analysis.

In this thesis, a large set of raw network data (>100Gb PCAP) will be provided to the student. The dataset includes benign network activities and dozens of real attacks performed by a red team during a large international cyber defense exercise. The goal of the thesis is to develop a method to make analyst's work more effective by including features such as network objects tagging, visualizing network activity tracking and identifying abnormal (malicious) activity patterns. The results of this thesis are expected to be used in a follow-up international cyber defense exercise in order to help the defense teams in identifying attacks as quickly as possible.

Bibliography

[1] L. S. 2018. Blue team 16 feedback.

[2] U. S. M. Academy. Cdx-2009 dataset. https://www.usma.edu/crc/sitepages/datasets.aspx.

[3] Apache. Apache kafka. https://kafka.apache.org.

[4] Apache. Apache storm. http://storm.apache.org.

[5] Apache. Metron. https://cwiki.apache.org/confluence/display/METRON/About+Metron.

[6] D. Arthur and S. Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[7] A. Ashari, I. Paryudi, and A. M. Tjoa. Performance comparison between naïve bayes, decision tree and k-nearest neighbor in searching alternative design in an energy simulation tool. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 4(11), 2013.

[8] R. A. R. Ashfaq, X.-Z. Wang, J. Z. Huang, H. Abbas, and Y.-L. He. Fuzziness based semi-supervised learning approach for intrusion detection system. *Information Sciences*, 378:484–497, 2017.

[9] R. Bace et al. An introduction to intrusion detection & assessment. *ICSA intrusion detection systems consortium white paper*, pages 1–38, 1999.

[10] R. Battiti. Using mutual information for selecting features in supervised neural net learning. *IEEE Transactions on neural networks*, 5(4):537–550, 1994.

[11] J. Benesty, J. Chen, Y. Huang, and I. Cohen. Pearson correlation coefficient. In *Noise reduction in speech processing*, pages 1–4. Springer, 2009.

[12] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita. Network anomaly detection: methods, systems and tools. *IEEE communications surveys & tutorials*, 16(1):303–336, 2014.

[13] C. M. Bishop. Pattern recognition and machine learning.

[14] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[15] A. Bronshtein. A quick introduction to k-nearest neighbors algorithm. https://medium.com/@adi.bronshtein/a-quick-introduction-to-k-nearest-neighbors-algorithm-62214cea29c7.

[16] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[17] R.-C. Chen, K.-F. Cheng, Y.-H. Chen, and C.-F. Hsieh. Using rough set and support vector machine for network intrusion detection system. In *Intelligent Information and Database Systems, 2009. ACIIDS 2009. First Asian Conference on*, pages 465–470. IEEE, 2009.

[18] Cisco. Netflow. https://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html.

[19] Cisco. Sourcefire intrusion prevention system. https://www.cisco.com/c/en/us/services/acquisitions/sourcefire.html.

[20] B. Claise. Specification of the ip flow information export (ipfix) protocol for the exchange of ip traffic flow information. Technical report, 2008.

[21] Cooperative Cyber Defence Centre of Excellence (CCDCOE). *Cyber Defence Exercise Locked Shields - After Action Report*, 2013.

[22] DEFCON. Defcon ctf archive. https://www.defcon.org/html/links/dc-ctf.html.

[23] Elastic. Elasticsearch - a nosql database solution. https://www.elastic.co.

[24] Elastic. Elasticsearch ingest node reference. https://www.elastic.co/guide/en/elasticsearch/reference/current/ingest.html.

[25] Elastic. Elasticsearch reference. https://www.elastic.co/guide/en/elasticsearch/reference/.

[26] Elastic. Elasticsearch's python api. https://elasticsearch-py.readthedocs.io/en/master/.

[27] elastic. Scripted fields in elasticsearch. https://www.elastic.co/guide/en/kibana/6.2/scripted-fields.html.

[28] R. Entezari-Maleki, A. Rezaei, and B. Minaei-Bidgoli. Comparison of classification methods based on the type of attributes and sample size. *Journal of Convergence Information Technology*, 4(3):94–102, 2009.

[29] L. Ertöz, M. Steinbach, and V. Kumar. Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data. In *Proceedings of the 2003 SIAM international conference on data mining*, pages 47–58. SIAM, 2003.

[30] S. J. S. et al. Description of the kddcup99 features. http://kdd.ics.uci.edu/databases/kddcup99/task.html.

[31] Fireeye. Fireeye. https://www.fireeye.com.

[32] C. for Applied Internet Data Analysis (CAIDA). Caida data - overview of datasets. http://www.caida.org/data/overview.

[33] C. I. for Cybersecurity University of New Brunswick. Flowmeter feature extraction tool. https://github.com/ISCX/CICFlowMeter.

[34] L. M. Garcia. tcpdump tool. http://www.tcpdump.org.

[35] A. Gharib, I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani. An evaluation framework for intrusion detection dataset. In *Information Science and Security (ICISS), 2016 International Conference on*, pages 1–6. IEEE, 2016.

[36] A. K. Ghosh and A. Schwartzbard. A study in using neural networks for anomaly and misuse detection. In *USENIX security symposium*, volume 99, page 12, 1999.

[37] F. E. Heba, A. Darwish, A. E. Hassanien, and A. Abraham. Principle components analysis and support vector machine based intrusion detection system. In *Intelligent Systems Design and Applications (ISDA), 2010 10th International Conference on*, pages 363–367. IEEE, 2010.

[38] Q. Hu, M. R. Asghar, and N. Brownlee. Evaluating network intrusion detection systems for high-speed networks. In *2017 27th International Telecommunication Networks and Applications Conference (ITNAC)*, pages 1–6. IEEE, 2017.

[39] F. Iglesias and T. Zseby. Analysis of network traffic features for anomaly detection. *Machine Learning*, 101(1-3):59–84, 2015.

[40] A. K. Jain and R. C. Dubes. Algorithms for clustering data. 1988.

[41] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323, 1999.

[42] H. G. Kayacik, A. N. Zincir-Heywood, and M. I. Heywood. Selecting features for intrusion detection: A feature relevance analysis on kdd 99 intrusion detection datasets. In *Proceedings of the third annual conference on privacy, security and trust*, 2005.

[43] T. M. Kodinariya and P. R. Makwana. Review on determining number of cluster in k-means clustering. *International Journal*, 1(6):90–95, 2013.

[44] A. H. Lashkari, G. Draper-Gil, M. S. I. Mamun, and A. A. Ghorbani. Characterization of tor traffic using time based features. In *ICISSP*, pages 253–262, 2017.

[45] W. Li, P. Yi, Y. Wu, L. Pan, and J. Li. A new intrusion detection system based on knn classification algorithm in wireless sensor network. *Journal of Electrical and Computer Engineering*, 2014, 2014.

[46] Z. Li, Y. Li, and L. Xu. Anomaly intrusion detection method based on k-means clustering algorithm with particle swarm optimization. In *Information Technology, Computer Engineering and Management Sciences (ICM), 2011 International Conference on*, volume 2, pages 157–161. IEEE, 2011.

[47] Y. Liao and V. R. Vemuri. Use of k-nearest neighbor classifier for intrusion detection1. *Computers & security*, 21(5):439–448, 2002.

[48] M. Liotine. *Mission-critical network planning*. Artech House, 2003.

[49] S. C. LLC. Cobalt strike 3.11 manual.

[50] G. Louppe, L. Wehenkel, A. Sutera, and P. Geurts. Understanding variable importances in forests of randomized trees. In *Advances in neural information processing systems*, pages 431–439, 2013.

[51] U. Mansoor. Cluster analysis using k-means explained. `https://codeahoy.com/2017/02/19/cluster-analysis-using-k-means-explained/`.

[52] Matplotlib. Matplotlib boxplots. `https://matplotlib.org/api/_as_gen/matplotlib.pyplot.boxplot.html`.

[53] J. Mazel, P. Casas, R. Fontugne, K. Fukuda, and P. Owezarski. Hunting attacks in the dark: clustering and correlation analysis for unsupervised anomaly detection. *International Journal of Network Management*, 25(5):283–305, 2015.

[54] J. McHugh. Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Transactions on Information and System Security (TISSEC)*, 3(4):262–294, 2000.

[55] R. Meier. Reconstructing https activity from traffic metadata. 2014.

[56] D. Meyer and F. T. Wien. Support vector machines. *R News*, 1(3):23–26, 2001.

[57] NETRESEC. Networkminer. `https://www.netresec.com`.

[58] M. Nicolau, J. McDermott, et al. A hybrid autoencoder and density estimation model for anomaly detection. In *International Conference on Parallel Problem Solving from Nature*, pages 717–726. Springer, 2016.

[59] U. of Bergen. Kddcup99 feature extraction tool. `https://github.com/AI-IDS/kdd99_feature_extractor`.

[60] C. C. D. C. of Excellence (CCDCOE). Locked shields. `https://ccdcoe.org/locked-shields-2017.html`.

[61] S. Onion. Security onion intrusion detection system. `https://securityonion.net`.

[62] T. M. Oshiro, P. S. Perez, and J. A. Baranauskas. How many trees in a random forest? In *International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pages 154–168. Springer, 2012.

[63] OSSEC. Ossec hids. https://www.ossec.net.

[64] W. Park and S. Ahn. Performance comparison and detection analysis in snort and suricata environment. *Wireless Personal Communications*, 94(2):241–252, 2017.

[65] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[66] B. project. Bro documentation. https://www.bro.org/sphinx/scripts/base/protocols/conn/main.bro.html.

[67] B. project. The bro network security monitor. https://www.bro.org.

[68] RAPID1. metasploit penetration testing framework. https://www.metasploit.com/.

[69] M. Roesch et al. Snort: Lightweight intrusion detection for networks. In *Lisa*, volume 99, pages 229–238, 1999.

[70] J. Ryan, M.-J. Lin, and R. Miikkulainen. Intrusion detection with neural networks. In *Advances in neural information processing systems*, pages 943–949, 1998.

[71] F. Sander, M. Ester, and P. Kriegelh. The algorithm gdbscan and its application. *Data Mining and Knowledge Discovery*, 2:178–192, 1998.

[72] Sguil. Sguil: The analyst console for network security monitoring. https://bammv.github.io/sguil/index.html.

[73] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *Proceedings of fourth international conference on information systems security and privacy, ICISSP*, 2018.

[74] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *computers & security*, 31(3):357–374, 2012.

[75] sklearn. Randomforestclassifier class. http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

[76] snort.org. Snort ids. https://www.snort.org.

[77] snort.org. Snort public rulesets. https://www.snort.org/downloads/#rule-downloads.

[78] snort.org. Snort's csv output plugin. https://www.snort.org/faq/readme-csv.

[79] R. Soltani, D. Goeckel, D. Towsley, and A. Houmansadr. Towards provably invisible network flow fingerprints. *arXiv preprint arXiv:1711.10079*, 2017.

[80] J. Song, H. Takakura, and Y. Okabe. Description of kyoto university benchmark data. *Available at link: http://âĂŃ www. âĂŃ takakura. âĂŃ com/âĂŃ Kyoto_âĂŃ data/âĂŃ BenchmarkData-Description-v5. âĂŃ pdf.[Accessed on 15 March 2016]*, 2006.

[81] Squertproject. Squert. http://www.squertproject.org.

[82] Suricata. Suricata intrusion detection system. https://suricata-ids.org.

[83] C. T. Symons and J. M. Beaver. Nonparametric semi-supervised learning for network intrusion detection: combining performance improvements with realistic in-situ training. In *Proceedings of the 5th ACM workshop on Security and artificial intelligence*, pages 49–58. ACM, 2012.

[84] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani. A detailed analysis of the kdd cup 99 data set. In *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*, pages 1–6. IEEE, 2009.

[85] tcpdump. Pcap fileformat. `https://www.tcpdump.org/pcap/pcap.html`.

[86] M. Usama, J. Qadir, A. Raza, H. Arif, K.-L. A. Yau, Y. Elkhatib, A. Hussain, and A. Al-Fuqaha. Unsupervised machine learning for networking: Techniques, applications and research challenges. *arXiv preprint arXiv:1709.06599*, 2017.

[87] Wireshark.org. Tshark manpage. `https://www.wireshark.org/docs/man-pages/tshark.html`.

[88] K. Wong, C. Dillabaugh, N. Seddigh, and B. Nandy. Enhancing suricata intrusion detection system for cyber security in scada networks. In *Electrical and Computer Engineering (CCECE), 2017 IEEE 30th Canadian Conference on*, pages 1–5. IEEE, 2017.

[89] C. Wurm. Analyzing network packets with wireshark, elasticsearch, and kibana. `https://www.elastic.co/blog/analyzing-network-packets-with-wireshark-elasticsearch-and-kibana`.

[90] J. Zhang and M. Zulkernine. A hybrid network intrusion detection technique using random forests. In *Availability, Reliability and Security, 2006. ARES 2006. The First International Conference on*, pages 8–pp. IEEE, 2006.

[91] J. Zhang, M. Zulkernine, and A. Haque. Random-forests-based network intrusion detection systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(5):649–659, 2008.

[92] X. Zhu. Semi-supervised learning literature survey. 2005.

[93] X. Zhu and A. B. Goldberg. Introduction to semi-supervised learning. *Synthesis lectures on artificial intelligence and machine learning*, 3(1):1–130, 2009.

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Network Monitoring and Attack Detection

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Känzig	Nicolas Luca Ermano Max

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zürich, 3.9.2018	NKänzig

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.