Traffic microstructures and network anomaly detection

Subtitle

Henry Clausen

A thesis presented for the degree of
Doctor of Philosophy

School of Informatics
University of Edinburgh
United Kingdom

June 22, 2021

Traffic microstructures and network anomaly detection

Subtitle

Henry Clausen

Abstract

Lorem ipsum dolor sit amet, consectetur adipisici elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquid ex ea commodi consequat. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisici elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquid ex ea commodi consequat. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisici elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquid ex ea commodi consequat. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisici elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquid ex ea commodi consequat. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisici elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquid ex ea commodi consequat. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

THE UNIVERSITY *of* EDINBURGH
informatics

Dedication

To mum and dad

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified. Some of the material used in this thesis has been published in the following papers:

- ....

Henry Clausen

Acknowledgements

I want to thank...

Contents

Chapter 1

Introduction

A significant threat to today's computer networks are attacks that aim to gain unauthorised access to sensitive infrastructure and information, especially as the increasing rate of zero-day attacks [?] threatens the traditional model of signature-based network intrusion detection. Such attacks are used to gain control or access information on remote devices by exploiting vulnerabilities in network services, and are involved in many of today's data breaches [?]. Anomaly-based intrusion detection methods are aimed to decrease the threat of zero-day attacks by not relying ... on libraries of known attack signatures, but their success is currently restricted to the detection of high-volume attacks using aggregated traffic features. Recent evaluations show that the current anomaly-based network intrusion detection methods fail to detect remote access attacks reliably [?].

Prominent network intrusion detection methods as Kitsune [?] or DeepCorr [?] learn structures in the sizes, flags, or interarrival times (IATs) of packet sequences for decision-making. These structures reveal information about attack behaviour, but are also influenced by a number of other factors such as network congestion or the transmitted data. We define **traffic microstructures** as reoccurring patterns in the metadata of packet sequences (such as used in shallow packet inspection) within a connection, such as the packet sizes of a Diffie-Hellman exchange or typical IATs of video streaming, or patterns within the summary statistics of individual connections or short sequences observed on a host such as the pattern of port 53 (DNS) connections being followed by port 80 or 443 connection (HTTP/S). No effort has been made so far to monitor or control these factors to probe these models for specific microstructures, with the current quasi-benchmark NID-datasets pay more attention to the inclusion of specific attacks and topologies rather than the documentation of the generated traffic. This situation has so far led researchers to often simply evaluate a variety of ML-models on these datasets in the hope of edging out competitors, without understanding model flaws and corresponding data structures through targeted probing. Sommer and Paxson citation have already should this be in?

The aim of this thesis is to improve the understanding of traffic microstrucutres and their shaping, and to define and propose methodologies to develop machine-learning based anomaly detection methods that leverage traffic microstructures effectively.

1.1 Central Research questions

Research Question 1
How well-structured is the space of microstructures observed in the traffic of a machine or a network? To what degree are these microstructures a result of specific computational activities that are of interest for traffic classification and intrusion detection, and how much are they affected by other external variables?

Research Question 2
To what degree can relevant microstructures in network traffic be captured in a model from a training dataset, and how can we achieve this? How can a model adapt to changes of structures in benign traffic?

Research Question 3
What is a meaningful representation of traffic microstructures? What requirements must a labelled traffic generation framework fulfill to provide realistic data?

Research Question 4
What will a contextual model be able to prevent?

1.2 Motivation

This section motivates the research on traffic microstructures and corresponding models and generation processes.

1.2.1 The case for machine learning and anomaly-based intrusion detection

Intrusion detection can be seen as a never ending arms-race between malicious actors on the one side that aim to access, manipulate, or damage computation infrastructure in a network, and the network operators, detection system providers and research community, on the other. Traditionally the most common approach to network intrusion detection is based on detecting closely defined notions of attacks, known as *attack signatures*. Since the late 1980's, signature based systems such as Snort or Bro citation have dominated the field of network intrusion detection due to high effectiveness, low false positive, and good computational efficiency. Due to their design, signature-based methods can only alert on known issues that had been categorised as threats on a signature list; zero-day attacks remain a large vulnerability of traditional IDSs.

In the 1990's, anomaly-detection emerged as a complementary detection tool by training machine learning or statistical analysis on large traffic datasets in order to detect variants on existing attacks or entirely new classes of attacks. However, inconsistent detection results and high false positive rates lead many administrators to believe machine-learning based intrusion detection to be unreliable, computationally too expensive, and headed for a slow death. In 2010, Sommer and Paxson discuss a number of reasons why machine-learning based detection methods failed to provide similar levels of success as their signature-based counterpart.

However, ...

Machine-learning and anomaly-based intrusion detection solutions are finding more widespread industrial application in products from companies such as Crowdstrike, Darktrace, citation. Furthermore, the rise of cloud computing has shifted the computational burden of complex intrusion detection systems from the monitored hosts to the cloud.

SPI and microstructures

Many ISP solutions rely on *deep packet inspection* (DPI), where payloads of packets are scanned for attack signatures or anomalies. While this offers a direct view at the content of communications, it raises several problems: The computational overhead of DPI is large and scales directly with the amount of transferred data. Furthermore, DPI infringes user privacy when scanning the content of messages, potentially allowing the operator to access the vast amount of personal information sent over the internet. Finally, DPI is incapable of processing encrypted traffic, an issue that malicious actors increasingly exploit. Some IDS-providers have started to decrypt any transferred traffic in a man-in-the-middle manner, but the ethical legitimacy of such approaches are at the very least controversial.

Shallow packet inspection (SPI) in contrast only inspects packet headers and is therefore computationally less expensive as well as robust against encryption and to some degree application evolution citation. SPI is used both by rule- and signature-based systems such as Snort, as well as by an ever increasing number of machine-learning based methods that attempt to make broad generalities about traffic solely from evaluating packet headers. insert some statistic how many.

For this, many traffic classification and anomaly-detection approaches leverage small-scale structures visible in the sizes, interarrival times, flags, port, etc. of packet sequences, which we call *traffic microstructures*. insert image and example here.

However, there exists no systematic description of these structures, and their usage is mostly only justified from high-level argumentation without any examination about the influences on them or their prevalence in the respective datasets used for evaluation. some additional arguments about model development?

1.2.2 Lack of model development in NID

1.2.3 Dataset problems and lack of labels

Newer models clearly leverage microstructures

However, traffic opaque

...

Relevance

1.3 Thesis overview

Chapter 2

Background

2.1 Network traffic

Computers in a network mostly communicate by sending *network packets* to each other, in which the transmitted information is encapsulated. Each network packet is split into the control information, also called packet header, and the user information, called payload. The packet header contains the necessary information for the correct transmission of the packet, including the transmission protocol layer (such as TCP, UDP, or ICMP), and the source and destination IP addresses and network ports[1]. It furthermore contains error-checking fields suchs as the size of the packet and a checksum over the payload, and protocol-specific fields.

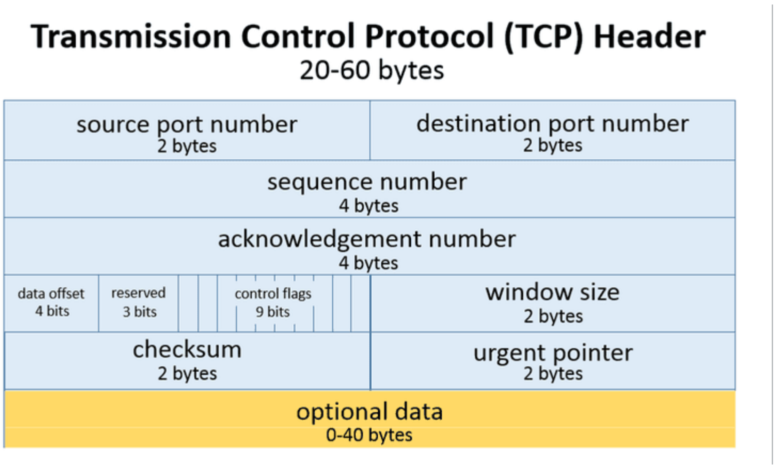


Figure 2.1: Typical format of a TCP packet. Source: https://www.lifewire.com/tcp-headers-and-udp-headers-explained-817970

The payload of a packet is in general the data that is carried on behalf of an application. It can be of variable length, however, it cannot exceed the maximum packet length set by the network protocol. In contrast to the packet header, the payload can be encrypted, a technique that makes it unreadable to any third parties and that is becoming increasingly common in modern computer networks.

While traversing between to parties, a network packet can pass multiple connecting devices which direct the packet in the right direction. Any device in

[1] A network port is a number that identifies which service or application is responsible for the processing of incoming packets.

the immediate circuit traversed by a packet can capture and store it. In a monitoring setting, packets are usually captured by network routers and stored in the widespread *pcap* format. In case of space shortage or privacy concerns, the payload of a packet can be dropped in the saving process.

Another more structured way of capturing network traffic is based on connection summaries, or **network flows**. RFC 3697 [?] defines a network flow as a sequence of packets that share the same source and destination IP address, IP protocol, and for TCP and UDP connections the same source and destination port. A network flow is usually saved containing these informations along with the start and duration of the connection and the number of packets and bytes transferred in the connection.

```
Date flow start          Duration Proto  Src IP Addr:Port       Dst IP Addr:Port   Packets  Bytes Flows
2010-09-01 00:00:00.459     0.000 UDP    127.0.0.1:24920    ->  192.168.0.1:22126        1     46     1
2010-09-01 00:00:00.363     0.000 UDP    192.168.0.1:22126  ->  127.0.0.1:24920          1     80     1
```

Figure 2.2: A typical network flow output

Raw packets grant full information about the connection, but take a lot of space when stored, whereas network flows give a more structured and lightweight overview over the traffic in a network. Both data formats are used widely by NIDSs.

2.1.1 Network attacks

Sophisticated data breaches affect hundreds of million customers and inflicts tremendous financial, reputational, and logistic damage. One reason for the recent rise of cyber crime is the increased use of sophisticated techniques for the attack of specific targets. Attackers use customised social engineering and custom-build malware that penetrate defensive barriers and potentially stay undetected in an infected system for an extended period of time.

In 1980, James P. Anderson, a member of the *Defense Science Board Task Force on Computer Security* at the U.S. Air Force, published the first report to introduce the notion of automated intrusion detection [?]. In it, he defines an **intrusion attempt** or a threat as

> *"... an unauthorized and deliberate attempt to access or manipulate information, or to render a system unreliable or unusable."*

Such attacks can be very diverse in their nature: They can be used to achieve different goals, and correspondingly exploit different types of tools and vulnerabilities. Very often, intrusive attacks involve some sort network communication between the victim machine(s) and a malicious agent. As this work is focusing primaly on network intrusion and corresponding defense systems, we will take a closer look at this type of communication. A recent survey covering intrusive attacks and defense systems distinguishes five classes of malicious network traffic [?]:

1. *DoS-attacks*: A denial-of-service attack is an attempt to remove ability of a particular computer to communicate with other machines over an extended

period of time. Such attacks are usually targeted at network servers in order to disrupt the service it is providing. All major types of DoS-attacks achieve this by overwhelming the target server with service requests, which are usually corrupted in a way that causes the server to bind resources unnessecarily long for each request, and thus losing its capability to process other requests. The most prominent type of DoS attacks are SYN-floods. They exploit the TCP-handshake protocol by sending many SYN-requests to a server while ignoring the SYN-ACK response packets sent in return by the server. This causes the server to keep waiting for a response for each of the attacker's requests, and thus binds the resources of the server while being computationally very cheap for the attacker. After a certain treshold, the server will not be able to process any more requests, rendering it unusable for actual client requests.

2. *Network probing/Reconnaissance attack*: The purpose of network probing attacks is to gather information about computers in a network and possibly find vulnerabilities which can be exploited in further attacks. This typically involves sending specific service requests to other computers in the network in order to gather information about this system, such as open ports or the operating system running on a machine, contained in the corresponding response packets.

   A common type of network probing attacks is *port scanning.* Its aim is to gather knowledge of computers in the network than run vulnerable services, such as HTTP servers, mail servers, and so on. A port scan achieves this by sending queries to one or more network ports on one or more computers in the network. A computer on which the contacted network port is open will respond to the query and thus reveils himself. A port scan can either be vertical, during many ports on one computer are scanned, or horizontal, where the attacker scans a small number of ports on many computers in the network.

   Network probing is often an integral part in the spreading mechanism of*computer worms.*

3. *Access Attacks*: These are attacks that attempt to gain unauthorized access to a machine. This could both be an individual from outside gaining access to the network, or a user from inside the network accessing services or privileges outside of their authority. Access attacks are often divided into *Remote-to-Local* (R2L) where a remote attacker gains access on a system over the network, and *User-to-Root* (U2R), where a user illegally gains administrator access to a machine. However, many attacks fall into both categories.

   Access attacks can be very diverse in their nature. A simple example are brute-force attacks where an attacker guesses the passwort of a user over a network service such as SSH. Other prominent and more sophisticated cases include *SQL injections*, where nefarious SQL statements are passed to an entry field for execution, or *buffer overflow*, in which more data is put into a buffer of a service than it can hold in order manipulate data in the memory past the buffer.

4. *Data Manipulation Attack*: Also known as "man-in-the-middle", these attacks typically involve an attacker reading and manipulating information in a data stream that is not addressed to him by exploiting vulnerable or missing authentication mechanisms in the IP protocol and related applications. A common form of such an exploit is *IP spoofing* where an attacker pretends to be a trusted computer by sending packets with a spoofed trusted source IP address. Similarly, vulnerabilities in digital certificates that serve as a unique identifier of a trusted computer can be used to create fake certificate and thus trick a victim into trusting the intruder carrying the faked certificate.

   Two examples of data manipulation attacks are *session replay* and *website impersonation*. In a session replay, the intruder captures packet sequences exchanged between two parties and modifies part of the data before forwarding it to the receiver. Here, both parties are unaware of the data manipulation and trust the authenticity of the connection. In website impersionation, a user is unknowingly redirected to a perfect copy of the website he requested. The user is then tricked to enter confidential information into a web form, which is then sent to the attacker instead of the trusted party operating the original page.

   Data manipulation attacks are often used pass malicious code to a victim in order to gain access on its machine. An impressive example of such behaviour was demonstrated by the malware *Flame*[2]: An infected host in a network sends messages to other machines running Windows advertising itself as a Windows update provider, using spoofed IP addresses and a fake Microsoft certificate and thus defeating Microsoft's authentication mechanism. Other computers were consequently tricked into receiving malicious updates from the infected host, which would then infect their machine [**?**].

5. *C&C traffic*:

   C& stands for *"Command and Control"* and denotes the communication between an infected host and a rogue agent, called the C&C server. The data transmitted in a C&C channel is usually exfiltrated information about the environment of the infected host, or commands from the C&C server for the victim for further operations. Typically, C&C communication is used for the control of one or more so called *bots*, computers that can perform tasks such as network scanning, establishing connections to other machines, or participating in DoS attacks. The communication between a bot and the C&C server is therefore extended and continuous over time. However, C&C communication can also be used for the request and transmission of an encryption key needed in a ransomware attack, in which it is limited in time and size.

   C&C communication is usually sent over the HTTP or HTTPS protocol as it is widely available and allows the attacker to hide their communication in the large volume of diverse traffic sent over this channel [**?**].

An additional class of networking threats is the *unauthorized surveillance* of

[2] also known as *Skywiper*

network traffic. A local network of computers is usually separated from the outside, with a router establishing the connection between computers in the network and ones outside the network. The traffic exchanged between machines inside the network therefore does not leave the network and is not visible for outsiders. Unauthorised access captures of internal network traffic can give an intruder significant information about the network topology, and even access to sensitive information if a connection is not encrypted. As network surveillance usually does not leave any visible traces in the network, I did not include it in the above listed types of malicious traffic.

2.2 Machine learning and anomaly detection

2.2.1 Sequence modelling

2.2.2 Classification vs anomaly-detection

Supervised vs self-supervised

2.2.3 Self-supervised language models

2.3 Intrusion detection

Sophisticated data breaches affect hundreds of million customers and inflict tremendous financial, reputational, and logistic damage. One reason for the recent rise of cyber crime is the increased use of advanced techniques for the attack of specific targets. Attackers use customised social engineering and custom-built malware that penetrate defensive barriers and potentially stay undetected in an infected system for an extended period of time.

Cyber-security relies on a range of defensive techniques, including sophisticated intrusion detection systems and firewalls that try to detect and prevent attacks against software subsystems. Malicious software still remains the biggest threat to computer users, and its detection is of utmost importance.

The field of intrusion detection is concerned with the development of methods and tools that identify and locate possible intrusions in a computer network. An *intrusion detection system* (IDS) is typically a device or software application that detects malicious activity or policy violations in an automated way by scanning incoming data collected from one or more sources for patterns that a model or a set of rules classifies as potentially malicious.

Intrusion detection is a well researched area, with the first IDSs emerging in the late 1980's. Intrusion detection today comprises a variety of research areas in terms of different types of data sources, system architectures, detection sope, and so forth. Denning [?] in 1987 established the notion of two different types IDS implementations: a) host-based and b) network-based. *Host-based intrusion detection systems* (HIDS) monitor each host in a network individually, and rely on local data streams such as application logs or raw operating system calls as data source. *Network-based intrusion detection* refers to the detection of malicious traffic in a network of computers and uses network traffic captures as a data source. In this PhD project, I will mainly focus on network traffic as a data source due its universal nature, its resilience against modification, and my previous experience in the field. However, I will also investigate the possibility of relating both types of traffic. For more details on IDS implementations, I refer the reader to my more detailed literature review.

Current detection methods are predominantly based on the analysis of previously identified attack signatures, which provides great reliability and low false alert rates. However, these methods are dependent on an updated attack signature database and provide no protection against previously unseen attacks. With attackers having access to more resources than ever, custom built malware is becoming more common to circumvent signature-based detection.

Another approach that has recently gained traction in commercial deployment is based on detecting malware and other undesired activity as anomalous behaviour when compared to benign computer activity. In this approach, known as **network anomaly detection** after Denning [**?**], models must quantify the behaviour of normal network behaviour to be independent of a narrowly defined notion of malicious activity.

2.3.1 Intrusion Detection Systems

The field of intrusion detection is concerned with the development of methods and tools that identify and locate possible intrusions in a computer network. An *intrusion detection system* (IDS) is typically a device or software application that detects malicious activity or policy violations in an automated way by scanning incoming data collected from one or more sources for patterns that a model or a set of rules classifies as malicious.

Intrusion detection is a well researched area, with the first IDSs emerging in the late 1980's. Intrusion detection today comprises a variety of research areas in terms of different types of data sources, system architectures, detection sope, and so forth. Figure **??** provides a broad yet uncomplete overview of these different areas.

Implementation

Denning [**?**] in 1987 established the notion of two different types IDS implementations: a) host-based and b) network-based. *Host-based intrusion detection systems* (HIDS) monitor each host in a network individually, and rely on local data streams such as application logs or raw operating system calls as data source. A relatively new form of host-based data sources are biometrik user data such as keystrokes or eye movement, please see [**?**] for more information.

Network-based intrusion detection refers to the detection of malicious traffic in a network of computers. A network intrusion detection system (NIDS) monitors network traffic within in a network and/or between the network and external hosts for malicious activity or policy violations. Network traffic data can take different forms, a more detailled explanation will be provided in Section 2.1.

Host-based systems have the advantage of working with high quality data that are typically very informative [**?**]. NIDS have the advantage of being plattform independent and more resiliant to attacks as detection of an infection is not done on the infected system. In this review, I will focus on work done in the area of network intrusion detection.

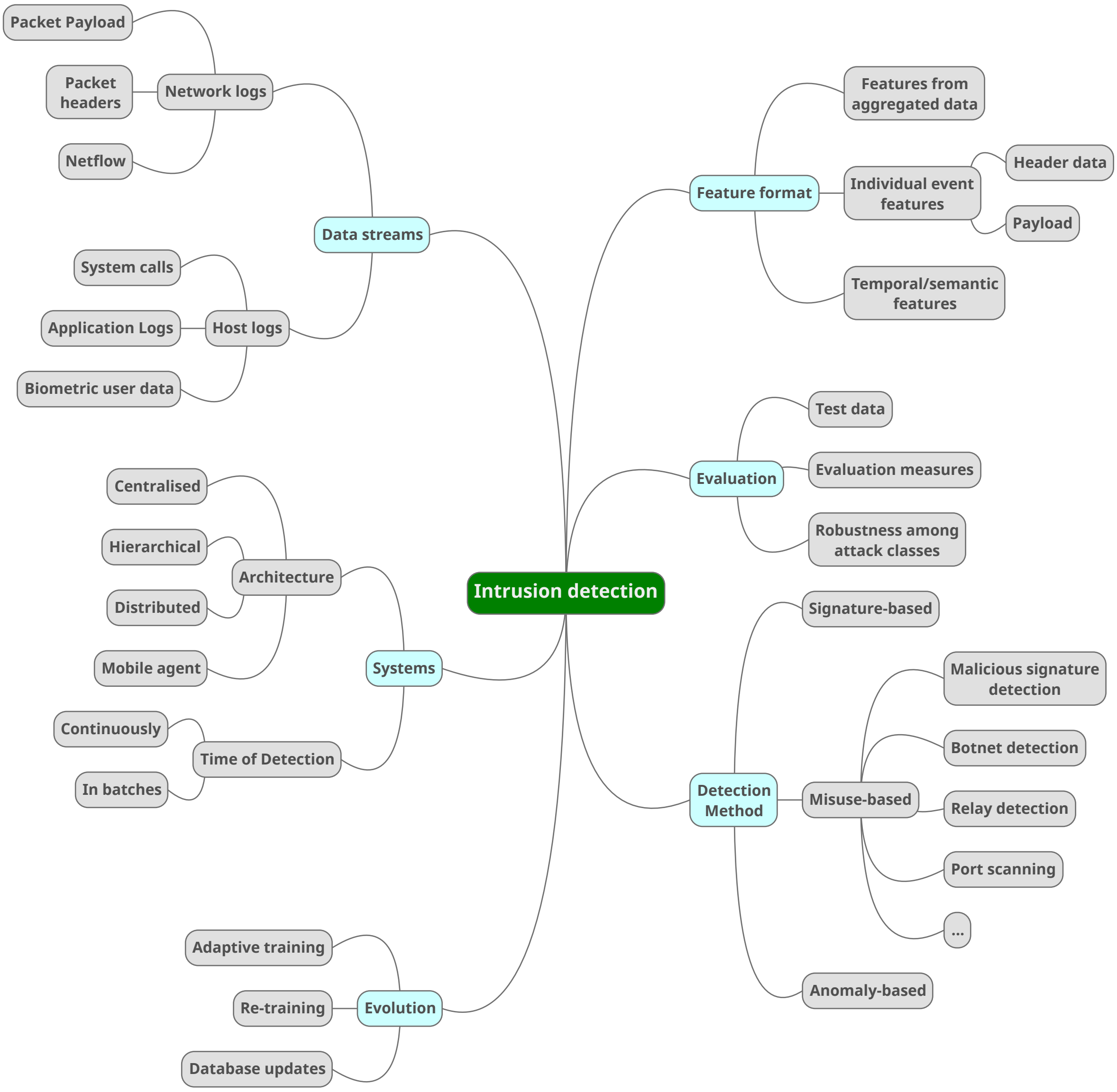


Figure 2.3: A broad overview over different aspects in an IDS

Detection methods

Detection methods are the core of an IDS, and are therefore the most important design choice. Traditionally, two broad types of detection approaches are identified: a) Misuse detection and b) anomaly detection.

Misuse detection aims at detecting a particular and well known reoccurring characteristic or pattern of a malicious behaviour. Two simple examples of such a characteristic are the large number of SYN packets sent by a host in a DoS attack, and the synchronised connection of many hosts to one server in a botnet. In misuse detection, abnormal or malicious behaviour is therefore defined first before developing a model to distinguish the defined behaviour from other traffic.

In contrast, anomaly detection aims at building a model of normal system behaviour that is accurate enough to spot any malicious behaviour as traffic that deviates from the estimated model. Anomaly detection is principally more difficult than misuse detection since the traffic model has to incorporate potentially very heterogenuous traffic behaviours. However, it is generally acknowledged that anomaly detectionhas is more suitable to detect new and previously unseen malicious behaviour as it makes no definite assumptions on the anomalous behaviour. Misuse detection is robust against evolution of malware as long as defined malicious behaviours do not change.

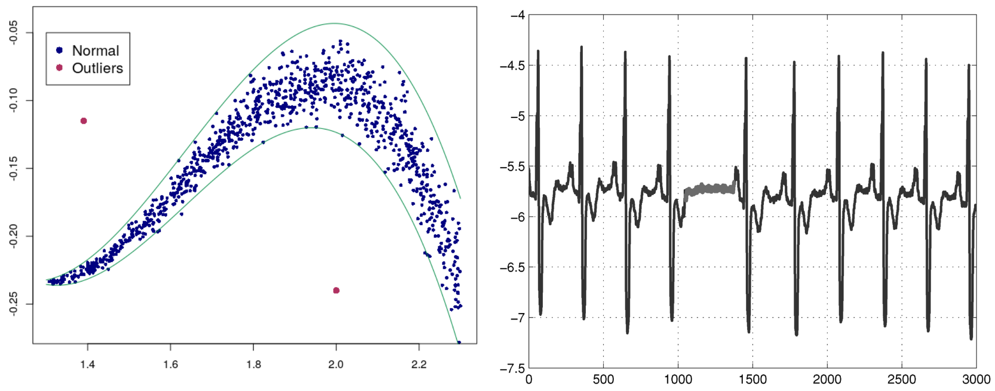
In reality, anomaly and misuse detection are not necessarily mutually exclusive, and there is a fluent passage between the two. This is because many anomaly detection approaches choose a particular set of features to be modelled with a particular threat in mind. For instance, models for the number of connections of a machine are naturally suitable for detecting DoS attacks, port scans, or Worm attacks.

As misuse detection detection methods are aimed at detecting very specific behaviour, they usually only detect one type of malicious traffic. Areas that have been researched particularly well include botnet C&C channel detection, DoS attack detection, and port scan detection. Areas that lack a comprehensive body of research are different types of R2L and U2R attacks. These are also currently the least detected attack classes [?].

2.3.2 Anomaly detection

Anomaly detection refers to the problem of identifying data instances that have significantly different properties than previously observed “normal data instances. Anomaly detection has its origins in statistical research where normal data instances are assumed to be generated as random variables by a probability distribution $P_N(X)$. New data is then identified as anomalous if its properties correspond to regions with vanishing likelihood, i.e. this particular data instance is highly unlikely to be generated by $P_N(X)$. Usually, the hard part in anomaly detection is to use observed data efficiently to build an estimated distribution $\hat{P}_N(X)$ that resembles $P_N(X)$ closely to identify anomalous events while asigning normal instances a non-vanishing likelihood. A variety of techniques exist to achieve this assuming comparably simple generating distributions. However, distributions for many interesting types of normal data can be complex and changing over time, and individual data points can have intricate interdependencies.

Anomaly detection has found wide application in areas where stability of



(a) (b)

Figure 2.4: The left plot (a) depicts simple anomalies that deviate in distance from regular data. The right plot (b) shows a contextual anomaly where a group of data instances do not follow a repeating timely behaviour with respect to all other data points (corresponding to an *Atrial Premature Contraction*, taken from [**?**])

critical systems or detection of their abuse is crucial. Examples of these include engine-fault detection in wind-turbines and fraud detection for credit cards. The assumption here is that the modelled system, such as the sensor-data stream of an engine or the buying behaviour of a customer, remains stable and generates consistent data. Detected anomalies in new observations then indicate a potential fault or abuse in this system. Obviously, it is not inherently clear that every abuse or fault generates data that differs from normal data. Therefore, it is important to choose data sources that are able to reflect the unique nature of a particular system.

Anomaly detection and NIDS

Anomaly detection has been applied to network security since the late 90's to assist IDSs. The behaviours of network computers are usually captured in the form of events logs such as network packets or flows, system calls, etc. The basic approach is to use collected training logs to infer a model of trustworthy activity, and then compare newly observed behaviour against this model. To prevent any anomaly model from learning malicious behaviour as normal, the used training data has to be free of any attacks[3].

As an example, an anomally large network flow involving an unusual pair of network computers could indicate that a hacked computer is extracting or sending out sensitive data to an unauthorized destination. An anomaly detection model does not make any assumptions about attack behaviour, but instead marks all deviations from the learned normal behaviour as potentially malicious. It is therefore better suited to find novel and unseen attacks. However, features mined from the data have to be able to reflect normal behaviour in a way that malicious traffic will look different in order to be detectable, which is not trivial.

[3] Again, the assumption is that unauthorized and malicious activity will correspond to behaviour that differs from trustworthy one.

Furthermore, normal instances that only occur rarely or are diffuse will potentially also be flagged as potentially malicious, making an anomaly-based IDS prone to high false alert rates.

2.4 Temporal correlation/Semantics-based

Despite network traffic being a stream of events, most anomaly-based intrusion detection approaches neglect any temporal features. Nevertheless, malicious behaviour most often is composed of a series of related computer and network events and have a distinct temporal and semantic profile [?]. As an example, an intruder using a session relay to exfiltrate information or pass malicious code generates a strong dependency between incoming packets in one connection and outgoing packets in the other connection (and vice versa) albeit the individual packets or connections resemble perfectly benign behaviour[4]. Building a model that can that can understand relations and dependencies between individual events could potentially lead to great improvements in the detection of otherwise non-anomalous attacks.

Since a machine's network traffic is the collective stream of multiple processes accessing the internet, individual traffic sequences are mixed with others, which makes the modelling of temporal dependencies a non-trivial task. In comparison to the other identified categories, research in the area of anomaly detection using temporal correlation or semantic models is sparse.

In 2003, **Krishnamurthy et al.** [?] propose a rather simple, yet efficient method for network-wide monitoring of individual key occurrences in an online fashion. For that, a sliding window approach is used to asign all keys (which here stand for individual IP addresses or network ports) a value containing the number of its occurrency. For each key, the collected values are then used to train either an *ARIMA* or a *Holt-Winters* method, both popular and powerful time-series forecasting models which can be trained in an online fashion. Anomalies are then identified as values with a forecasting error exceeding a certain treshold and thus indicating a sudden change in occurrency-behaviour of that particular key. This is an improvement to summarisation measures such as entropy or histograms in two ways: It provides a better resolution of individual traffic channels, enabling the detection of attacks with far less volume, and enabling the modelling of more complex temporal patterns which become apparent on a lower level. And it makes attack attribution far simpler by indicating directly the key which is subject to an anomaly.

Since traffic is usually arriving at a fast rate, it is a computationally hard and memory-consuming task to count the occurrencies of all keys simultaneously. Schweller et al. overcome this problem using a *sketch-based counting approach* which uses hash-function to direct the values of a key directly to a position in a hash table without the need to store actual key in the memory. As this process is not exact, the count value is subject to statistical variations and the authors propose an unbiased estimator. The inaccuracy of the count estimator is also preventing the authors from using a probabilistic approach to anomaly detection instead of simple tresholding. However, this approach is also only counting

[4] as they are essentially two normal connections that are relayed by the intruder

occurrencies and does not detect any correlated key behaviour.

A great problem of the proposed framework is the fact that the key translation works only in one direction, making it impossible to associate a detected change with the corresponding key. This problem is overcome by **Schweller et al.** [?] who propose a reversible sketch method.

Pellegrino et al. [?] last year proposed *BASTA*, a framework to mine behavioural fingerprints from network flows using *timed automata learning.* An automata encodes patterns of short-term interactions of a system and is a representation of symbolic sequences, often corresponding to state transitions, which it encodes in a state transition function. Applied to network flows, an automata represents sets of events[5] that can follow each other in the network trace of a system. To be more specific, they used a type of probabilistic automata that works with transition probabilities and thus works in a similar way to a *hidden Markov model.* Events are asigned a state corresponding to the protocol and the direction of the flow, and the quantile[6] its duration, size, and number of packets are lying in. The authors then use this framework to create malicious fingerprints by training it on malicious traffic intermixed with normal traffic. These fingerprints are then detected on an infected host if the difference between the expected and the observed state counts drops below a treshold.

This paper is itself not developing any anomaly detection techniques, and I will not discuss the flaws of its application for malware fingerprinting. It is interesting for us as the developed automata mining techniques can also find direct application in mining normal behaviour automata and thus be used in an anomaly detection model.

Noble and Adams [?, ?] have recently proposed *ReTiNa*, a tool that measures temporal changes in the correlation between individual events in order to find intrusions on individual hosts. In their approach, they estimate the correlation between the time passed between two events, also called *interarrival time*, of an OD pair and the associated size or number of packets of the involved events. For this, interarrival time and the size/packet number are modelled as a bivariate gaussian distribution, and the covariance matrix is estimated using maximum-likelihood-estimation. The authors use a sophisticated online-estimation method to adapt the estimates to changes in the correlation structure, which can then be identified by comparison to an offline estimate. Anomalies are then identified as a collection of changes happening across multiple OD pairs on one host or in the entire network by simple hypothesis testing, which decreases the false-positive rate. The assumption here is that different OD pairs are independent of each other.

A big advantage of this approach is that it is adaptive and does not need a training phase, i.e. it is not reliant on attack-free training data. The method was tested both on the LANL network flow data as well as internal data from the *Imperial College Academic network.* The method found several anomalies that coincide malicious activity in the network, but a definitive conclusion whether they are related is difficult to make.

Whitehouse, Evangelou and Adams [?] modeling the number of network flow and *user authentication* events on individual hosts as a polynomial function

[5] distinguished by port, protocol, etc.

[6] of the overall distribution of the individual parameters

of the time and day and its rarity. Anomalies are then identified using Fisher's product test statistic and the reconstruction error. The method was tested on the LANL data using the auth and the flow sources and wa able to identify persistent structures in the data.

2.4.1 Application to stepping stone detection

Especially in larger computer networks, attackers often try use relay-like command chains, also called *stepping stones*, to obfuscate their origin or access machines without external connection. In such a chain, no direct connection exists between the first and the last machine, commands are sent through one or multiple intermediate machines. Stepping stone connections are usually encrypted and are notoriously difficult to identify. Upon detection, pairs of stepping stones are a clear indication of anomalous activity.

As stepping stone detection falls much more in the field of misuse detection, I will only give a very brief overview over emplyed techniques.

Zhang and Paxson [?] proposed one of the first methods for detecting stepping stones in 2000. They model relayed key-stroke packet streams as a two-dimensional ON/OFF switching process, where state changes have to occur within a certain window between the two channels. Correlation is then simply detected if the number of switches lying in this window exceeds a treshold.

A common assumption made when trying to find stepping stones is that packet or flow streams between different hosts in a network are almost always independent of each other, which is shown by [?]. **He and Tong**[?] model normal packet arrivals in a connection as a Poisson Process, and use the assumption of independence to derive a probabilistic distribution for the similarity of packet numbers in two channels in a time interval. P-values are then used to identify when the number of similar intervals becomes unlikely. They also show that if the ratio of chaff-packets exceeds to necessary packets in a stepping stone becomes too large, correlation is impossible to detect under the asumption of normal traffic following a Poisson distribution.

Neil et al. [?] also model normal event arrivals along an edge as a *negative Binomial process* with two different rates, evolving as a hidden Markov model, and correlation in different time intervals in then used using a likelihood ratio test to obtain p-values. In their approach, instead of testing network wide correlation, which is computationally unfeasible, testing is done on two different forms of local subgraphs, a star shape and a path-shape on selected edges.

2.4.2 Semantic-based approaches using different data sources

Approachs that model semantic or temporal behaviour characteristics have been also been applied on host-based data streams such as *system call logs* or *process logs* as well. As these are mostly symbolic event streams, anomaly detection in principal works similarly as for network traffic. As these data sources have usually a more hierarchical structure of events following each other in a parent-child fashion and therefore do not suffer from overlapping signals, and it is generally easier to identify semantic structures. Notable examples include the use of deterministic automata [?], Markov chains [?], hidden Markov models [?, ?], or *recurrent neural networks*[?].

2.5 Existing datasets

In order to evaluate their ability to model the behaviour of a network and to identify malicious activity and network intrusions, new methodologies have to be tested using existing datasets of network traffic. This network should ideally contain realistic and representative benign network traffic as well as a variety of different network intrusions. However, as network traffic contains a vast amount of information about a network and its users, it is notoriously difficult to release a comprehensive dataset without infringing the privacy rights of the network users. Furthermore, the identification of malicious traffic in network traces is not straightforward and often requires a significant amount of manual labelling work. For that reason, only a handfull of datasets for network intrusion datasets containing real world traffic exist. There have also been some efforts to artificially creating such datasets and thus bypassing any privacy concerns. However, up to today, no artificial dataset truly resembles real network traffic in every aspect [?].

As described in a recent survey by **Ahmed et al.** [?], we can generally distinguish four different types of datassets containing network traffic:

1. **Real network data containing known intrusions**:
2. **Real network data containing injected intrusions**:
3. **Real network data containing no intrusions/untruthed real network data**:
4. **Synthetic network data with/without injected intrusions**:

I will now describe the properties of existing datasets suitable for network intrusion detection. As this review is primarily concerned with anomaly detection models that require benign network traffic, I did not include datasets such as honeypots that mostly contain malicious traffic. A description that includes such datasets can be found in the below described survey by Ahmed et al. [?].

Los Alamos National Laboratory, 2015 - Comprehensive, Multi-Source Cyber-Security Events [?][?]

In 2015, the Los Alamos National Laboratory (LANL) released a large dataset containing **network flow** traffic from teir corporate computer network, which contains about 17600 computers. The data was gathered over a period of 58 days with about 600 million events per day. The data only contains internal network connections, i.e. no flows going to or coming from computers outside the network are included. IPs and ports were de-identified (with the exception of the most prominent port), but are consistent throughout the data. Since the data stems exclusively from one corporate network, it can be assumed that it shows more homogeneity in the observed traffic patterns than general network traffic.

Additionally, the dataset also contains other event sources which were recorded in parallel in order to give a more comprehensive look at the network, and could be very useful when investigating a detection approach that correlates multiple event sources. These sources include process events and authentication events from Windows-based computers and servers, and DNS lookup events from the DNS servers within the network.

The dataset furthermore contains a labeled set of redteam events which should resemble intrusions. However, these events are not part of the network flow data and only contain information about the time of the attack and the attacked computer. These events apparently resemble remote *access attacks*, are not described further and appear to be artificial or injected into the dataset. It is thus not certain how well they resemble actual network intrusions.

LANL released another dataset containing network flow traffic from their network in 2017 [?]. This dataset is similar to the one from 2015, but spans over a longer period of time, 90 days. Furthermore, it contains no labeled malicious activity, however that does not mean that the data is completely free of malicious activity.

CTU 2013 [?, ?]

The *Stratosphere Laboratory* in Prague released this dataset in 2013 to study botnet detection. It consists of more than 10 million labeled **network flows** captured on lab machines for 13 different botnet attack scenarios. Additionally, the raw packets for the botnet activity is also available for attack analysis.

The labelling in this dataset is different from other datasets as each flow in the list is labeled based on the source IP address. In the experiments, certain hosts are infected with a botnet and any traffic arising from such a host is labeled as Botnet traffic. Traffic from uninfected hosts is labeled as Normal. All other traffic is Background, as one cannot classify it.

A criticism of this dataset is the unrealistically high amount of malicious traffic contained in the dataset, which makes it easier to spot it while reducing false positives. Furthermore, the way normal or background traffic is generated is described only poorly and leaves the question how representative it is of actual network traffic.

UGR 2016 [?]

The UGR'16 dataset was released by the University of Grenada and contains **network flow**[7] data from a spanish 3-tier ISP. This ISP is a cloud service provider to a number of companies, and thus the data comes from a much less structured network than the LANL data. It contains both client's access to the Internet and traffic from servers hosting a number of services. The data therefore contains a very wide variety of traffic patterns, an advantage emphasised by the authors. IP-adresses are consistently anonymised while network ports are unchanged. However, it is not ensured that the traffic capture is complete, i.e. that all traffic coming from and going to a particular machine is captured.

A main focus in the creation of the data was the consideration of long-term traffic evolution and observable periodicity in order to enable the testing of so called *cyclostationary* traffic models. The dataset correspondingly covers a very

[7] netflow v9

long period, spanning from March to August of 2016, and containing about 14 GB of traffic per week.

The data is split into a training set and a test set, with the latter containing labeled attack data. This attack data does not stem from rogue agents but is in part generated in controlled attacks on victim machines, and in part injected from previously observed malware infections. The attack data is therefore does not truly correspond to actual attacks, but achieves a high degree of similarity. The implemented attacks contain:

- DoS attacks (controlled attacks),
- Port scanning (controlled attacks),
- C&C traffic from a botnet (injected).

The authors also acknowledge that the background traffic is not necessarily free from further attacks. In fact, three real attacks have been observed and labeled, corresponding to IP-scanning and a spam mail campaign.

UNSW-NB 2015 [?]

The dataset realeased by the *University of New South Wales* in 2015 contains real background traffic and synthetic attack traffic collected at the "Cyber Range Lab of the Australian Centre for Cyber Security". The data is collected from a small number of computers which generate real background traffic, and is overlayed with attack traffic using the *IXIA PerfectStorm tool*. The time span of the collection is in total 31 hours.

An advantage of the collected dataset is the inclusion of both **raw packets** and **network flows** along with two other data formats containing newly engineered features. This allows a more detailed analysis of the data and possibly a better distinction between attack and benign traffic. In total, the data contains 260 000 events.

Another advantage of the data is the variety of attack data, containing a number of DoS, reconnaissance, and access attacks. However, due to the synthetical injection of these attacks, it is unclear how close they are to real-world attack scenarios.

Since this dataset is collected from a relatively small number of machines and during a limited period of time, it is furthermore unclear how suitable for capturing both the temporal evolution and the heterogeneity of real background traffic.

CICIDS 2017/2018 [?][?]

This dataset, released by the *Canadian Institute for Cybersecurity* (CIC), contains 5 days of network traffic from 12 computers. These computers all have either different different operating systems such as Windows, OSX, or Ubuntu, or different versions of the same operating system in order to enable a wider range of attack scenarios. The network furthermore contains switches, routers, a web server, a modem, and a firewall in order to ensure a realistic network topology. The traffic data itself consists of **labeled benign and attack traffic**, and is available as 11 GB per day of **raw packets** with payloads, or as **network flows**.

It was ensured that the data contains all traffic coming and going from individual machines. However, in contrast to other datasets, the background traffic is not directly generated through user interactions on the machine, but by using a method to profile abstract user behaviour in different traffic protocol. The purpose of this is to make the traffic more heterogenuous and to ensure that different types of behaviour are present in the data during the comparably short time span. This However, it is not completely clear how much of the underlying structure of real traffic is lost in the process, and therefore how suitable this data is to build models of benign user activity.

The attack data of this dataset is one of the most diverse among NID datasets, as it contains a variety of up-to-date attacks, such as different types of DoS attacks, SQL-injections and Heart-bleed attack, network scanning, or botnet activity. These are not always successful in order to reflect actual attack scenarios. However, the authors did not describe very well how the data from these attacks is generated and combined with the background traffic as it is also processed through a form of profiling engine.

CICIDS 2018: This dataset is generated in a similar fashion to the CICIDS 2017 data that we used in this work. The main differences are that the CICIDS 2018 data spans over three weeks and includes in total 450 hosts, but lacks the amount of web-attacks that we require and which is present in the CICIDS-2017 dataset.

The CIC released another very similar dataset to these ones in 2012.

DARPA 1998 [?]

The *Defense Advanced Research Projects Agency* released the first major dataset to test network intrusion detection systems. The data stems from two experiments at the *MIT Lincoln Laboratory* were multiple victim hosts running Unix and Windows NT were subject of over 200 attacks of 58 different types. The data spans three weeks of training and two weeks of testing data and contains *raw packets* that are labeled. It was since then heavily used as a benchmark to test new detection methods.

Also due to its prominence, it was heavily scrutinised and received a lot of criticism for its lack of realistic background traffic, which was generated through simulation procedure, and the presence of artifacts from these simulations in the data that could heavily skew any model relying on benign traffic. Also, the high percentage of attack traffic in the data is described as unrealistic.

Furthermore, since the dataset is now more than 20 years old, it is remarked that both the benign and attack traffic does not resemble modern network traffic anymore.

KDD Cup 1999 [?, ?]/NSL-KDD 2012 [?]

The *MIT Lincoln Laboratory* created this dataset in 1999 by processing portions of the 1998 DARPA dataset with new labels for a competition at the conference on *Knowledge Discovery and Data Mining*, and is the most widely used dataset in intrusion detection. It contains 2 million connections summaries in a new format and in total 38 attack types. This new format is essentially a form of **network flows** with a greatly increased number of features, 46 in total, which give additional details about the origin of the connection. The availability of these features in a real-world application however is in my opinion unrealistic as most of them could be mined due to the availability of parallel surveilance of the host, a Solaris-based system. However, we can see significant differences in today's operating systems which barely resemble Solaris. In addition, parallel host system surveilance usually cannot be taken for granted in a realistic network environment. Naturally, as the KDD'99 data stems directly from the DARPA dataset, it also faces the same problems and criticism.

The *Canadian Institute for Cybersecurity* postprocessed the KDD'99 data in order to address some of its shortcomings. This includes removing redundant records, balancing the size of the training and test data, and adjusting the proportion of attack traffic in the data. However, the biggest criticism from the KDD'99 and the DARPA data, the unrealistic generation of background data, still prevails.

LBNL 2013 [?]

This dataset released by the *Lawrence Berkeley National Laboratory* in 2005 is the first one to examine internal network traffic inside a modern enterprise. It contains more than 100 hours of *packet headers* from several thousand internal hosts.

This dataset contains no known attack traffic, and is therefore only suitable for traffic analysis and model fitting analysis. Furthermore, as being the first dataset containing enterprise traffic, privacy concerns caused the authors to remove any possibilities to identify individual IP addresses.

In 2011, Saad et al. [?] combined this dataset with existing botnet traffic to create a dataset containing both benign and attack traffic.

UNIBS 2009[?]

This dataset was collected on the campus network of the *University of Brescia* on three consecutive days in 2009. The dataset contains in total 79000 anonymised TCP and UDP *network flows.*

This dataset is not directed towards intrusion detection research, but was made as *ground truth data* for traffic classification. It therefore contains labels which indicate which of in total six applications generated the corresponding traffic flow. It might however still be of interest for model assessment in intrusion detection that is relying on traffic classification.

CAIDA 2016 [?]

The *Center for Applied Internet Data Analysis* started collecting network traces from a high-speed backbone link in 2008 with the collection still ongoing. The data is available in anonymised yearly datasets containing one hour of **packet headers** for each month.

Since the traffic is collected from a backbone link, it is very unstructured and heterogenuous. It is furthermore not necessarily free from attack traffic. Although this dataset has been used for intrusion detection before, it is more suitable for general internet traffic analysis.

MAWI 2000 [?]

Similarly to the CAIDA dataset, this dataset contains **packet headers** from the WIDE backbone. It is therefore similarly unstructured, anonymised, and not free from attack traffic. Since this dataset was already collected and released in 2000, it can also be remarked that the contained traffic is too old to represent modern traffic.

ADFA 2013/2014 [?, ?]

The ADFA datasets, released by the *University of New South Wales*, focuses on attack scenarios on Linux and Windows systems as well as **stealth attacks**. To create host targets, the authors installed web servers and database servers , which were then subject to a number of attacks.

The dataset contains both attack traffic and benign traffic. However, the dataset is directed more towards attack scenario analysis and is criticised as being unsuitable for intrusion detection due to its lack of traffic diversity. Furthermore, the attack traffic is not well separated from the normal one.

ICT datasets [?]

The *Impact Cyber Trust* releases cyber security oriented data. Its repository includes many datasets, synthetic as well as real captures, from different sources. Many datasets focus on observed attack data and thus are not directly applicable to intrusion detection. Furthermore, there is in general very little information provided that describes a dataset's origin, which makes it hard to investigate the network topology.

Among the more useful datasets are the *USC datasets*[8], which contain network traffic (both **packet headers** and **network flows**) from academic networks in the US between 2008 and 2010. The datasets are very large, with the largest one covering 48 hours and containing 357 GB of packet headers.

LITNET-2020 [?]

The LITNET-2020 from the Kaunas University of Technology Lithuania from 2020 was collected from an academic network over a timespan of 10 months and contains annotated real-life benign and attack traffic. The corresponding network provides a large network topology with more than a million IP-addresses, and the data was collected in the form of network flows with more than 80 features. However, the dataset only contains traffic from high volume attacks such as DoS-, scanning, or worm attacks, which are not suitable to evaluate CBAM.

Boğaziçi University distributed denial of service dataset

[?]

The Boğaziçi University distributed denial of service dataset contains both benign traffic from more than 400 users from an academic network as well as artificially created DoS-attack traffic. The dataset spans only 8 minutes and contains no access attacks.

[8] DS-062, LANDER Data, and DS-266

Chapter 3

Related work

3.1 Traffic examinations

- Paxson Sommer
  -

3.2 Traffic generation frameworks

3.3 Small-scale anomaly detection approaches

Chapter 4

Modelling

4.1 Controlling network traffic microstructures for machine-learning model probing

Network intrusion detection models increasingly rely on learning traffic microstructures that consist of pattern sequences in features such as interarrival time, size, or packet flags. We argue that precise and reproducible control over traffic microstructures is crucial to understand and improve NID-model behaviour. We demonstrate that probing a traffic classifier with appropriately generated microstructures reveals links between misclassifications and traffic characteristics, and correspondingly lets us improve the false positive rate by more than 500%. We examine how specific factors such as network congestion, load, conducted activity, or protocol implementation impact traffic microstructures, and how well their influence can be isolated in a controlled and near-deterministic traffic generation process. We then introduce DetGen, a traffic generation tool that provides precise microstructure control, and demonstrate how to generate traffic suitable to probe pre-trained NID-models.

4.2 Introduction

Scientific machine learning model development requires both **model evaluation**, in which the overall predictive quality of a model is assessed to identify the best model, as well as model validation, in which the behaviour and limitations of a model is assessed through targeted **model probing**, as depicted in Fig. 4.1. Model validation is essential to understand how particular data structures are processed, and enables researchers to develop their models accordingly. Data generation tools for rapid model probing such as the *What-If tool* [**?**] underline the importance of model validation, but are not suitable for providing probing data that resembles the complex structures found in network packet streams.

Machine-learning breakthroughs in many fields have been reliant on a precise understanding of data structure and corresponding descriptive labelling to develop more suitable models. In *automatic speech recognition (ASR)*, tone and emotions can alter the meaning of a sentence significantly. The huge automatically gathered speech datasets however only contain speech snippets and if possible their plain transcripts. While modern speech models are in principle able to

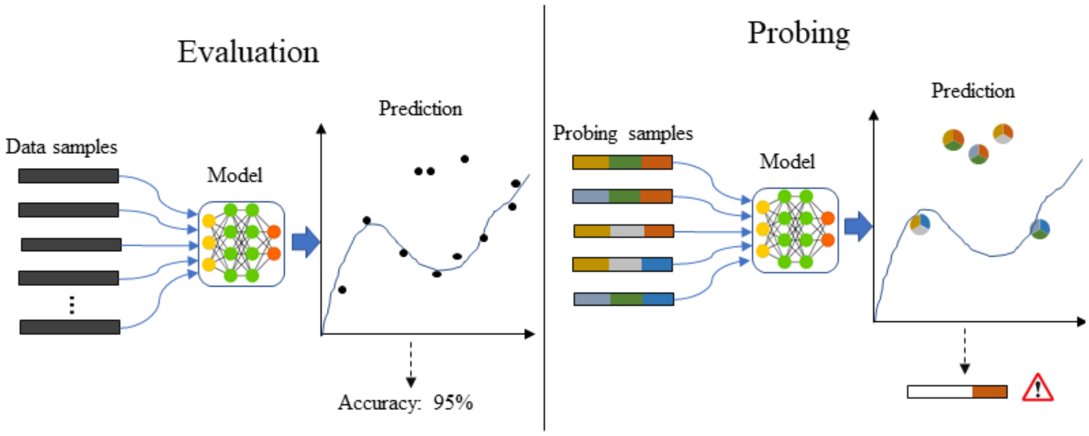


Figure 4.1: Model evaluation and model probing with controlled data characteristics.

learn implicit structures such as emotions without explicit labels, it is impossible to determine the cause for systematic error when they are not. Datasets that contain labelled specialised speech characteristics such as the Ryerson Database of Emotional Speech and Song (RAVDESS) [?] not only allow researchers to identify if their model is susceptible to structural misclassification through targeted probing, but also inspire new methods to capture and understand these implicit structures [?], which in turn leads to design improvements of general speech recognition models [?].

Prominent network intrusion detection methods as Kitsune [?] or DeepCorr [?] learn structures in the sizes, flags, or interarrival times of packets for decision-making. These **traffic microstructures** that can be observed when looking sequences of packets or connections reveal information about attack behaviour, but are also influenced by a number of other factors such as network congestion or the transmitted data. However, no effort has been made so far to monitor or control these factors to probe these models for specific microstructures, with the current quasi-benchmark NID-datasets pay more attention to the inclusion of specific attacks and topologies rather than the documentation of the generated traffic. This situation has so far led researchers to often simply evaluate a variety of ML-models on these datasets in the hope of edging out competitors, without understanding model flaws and corresponding data structures through targeted probing.

We demonstrate how to produce traffic effectively to probe a state-of-the-art traffic classifier, and why a certain degree of generative **determinism** is required for this to isolate the influence of traffic microstructures. The model insights and corresponding performance improvements achieved through probing motivate our experimental examination of various influence factors over microstructures and how to control them during traffic generation. Finally, we present *DetGen*, a traffic generation tool that provides near-deterministic control over microstructure-shaping factors such as conducted activity, communication failures or network congestion to generate reproducible traffic samples along with corresponding ground-truth labels.

This work provides the following contributions:

1. We demonstrate why model probing with controllable traffic microstructure is a crucial step to understand and ultimately improve model behaviour by probing a state-of-the-art LSTM traffic classifier and lowering its false positives five-fold.

2. We discuss requirements for traffic data suitable to probe models pre-trained on a given NIDS-dataset, and demonstrate how to generate probing traffic effectively through DetGen-IDS, a dedicated probing dataset.

3. We examine experimentally how different factors affect traffic microstructures, and how well they can be controlled in a more effective manner when compared to common VM-based traffic generation setups.

4. We propose DetGen, a container-based traffic generation paradigm that provides accurate control and labels over traffic microstructures, and experimentally demonstrate the level of provided generative determinism to traditional generation set-ups.

DetGen and the DetGen-IDS data are openly accessible for researchers on *GitHub*.

4.2.1 Outline

The remainder of the paper is organized as follows. Section 4.3 discusses the need for generating probing data with sufficient microstructure control before presenting the probing and corresponding improvement of a state-of-the-art intrusion detection model as a motivating example. Section 4.5 discusses how to generate probing data appropriately for pretained models, and provides an overview over the DetGen-IDS data. Section 4.6 proceeds to examine over which traffic characteristics DetGen exerts control and the corresponding control level. Section 4.7 provides details over the design paradigm of DetGen and the resulting advantages over traditional setups, while Section 4.8 discusses the level of control DetGen provides when compared to traditional setups. Section 7.6 concludes our work.

4.2.2 Existing datasets and corresponding ground-truth information

Real-world NID-datasets such as those from the Los Alamos National Laboratory [?] (LANL) or the University of Grenada [?] provide large amounts of data from a particular network in the form of flow records. Due to the lack of monitoring and traffic anonymisation, it is impossible for researchers to extract detailed information about the specific computational activity associated with a particular traffic sample. Synthetic NID-datasets such as the CICIDS-17 and 18 [?] or the UNSW-NB-15 [?] aim to provide traffic from a wide range of attacks as well as an enterprise-like topology in the form of `pcap`-files and flow-statistics. The CICIDS-17 data for example contains 5 days of network traffic from 12 host that include different Windows, Ubuntu, and Web-Server versions, and covers attacks from probing and DoS to smaller SQL-injections and infiltrations. While some effort is put in the generation of benign activities using activity scripting or traffic generators, we have seen no attention being spent at monitoring these activities accordingly, which leaves researchers with the limited information available through packet inspection. Furthermore, synthetic datasets can be criticised for their limited activity range, such as the CICIDS-17 dataset where more than 95% of FTP-transfers consist downloading the Wikipedia page for ‘Encryption’ [?], which leads to insufficient structural nuances to for effective probing.

4.2.3 Scope of DetGen

The scope of DetGen is to generate traffic with near-deterministic control over factors that influence microscopic packet- and flow-level structures. DetGen separates program executions and traffic capture into distinct containerised environments to exclude any background traffic events, simulates influence factors such as network congestion, communication failures, data transfer size, content caching, or application implementation.

4.3 Methodology and example

Assume the following problem: You are designing a packet-level traffic classifier which is generating a significant number of false positives, something that is still a common problem for the state-of-the-art [?]. The false positives turn out to be caused by a particular characteristic such as unsuccessful logins or frequent connection restarts. However, existing real-world or synthetic datasets do not contain the necessary information to associate traffic events with these characteristics, which prevents you from identifying the misclassification cause effectively. To address this problem, we need a way to controllably generate and label traffic microstructures caused by these characteristics.

To provide an example, we look at a *Long-Short-Term Memory* (LSTM) network, a deep learning design for sequential data, by Hwang et al. [?], which is designed to classify attacks in web traffic and has achieved some of the highest detection rates of packet-based classifiers in a recent survey [?]. Through probing we will learn that retransmissions in a packet sequence dramatically deplete the model's classification accuracy. We take the following steps:

Step 1: Determine model performance and feed it suitable probing traffic.

Step 2: Examine the correlation between traffic misclassification scores and the generated traffic microstructure labels to find a likely cause.

Step 3: Examine at which latency levels specific connections are misclassified.

Step 4: Generate two similar connections, with one exposed to strong packet latency.

Step 5: Show that by removing retransmission sequences in the preprocessing, misclassification is significantly reduced.

Step 1: To detect SQL injections, we train the model on the CICIDS-17 dataset [?] (85% of connections). For the evaluation, we also include a set of

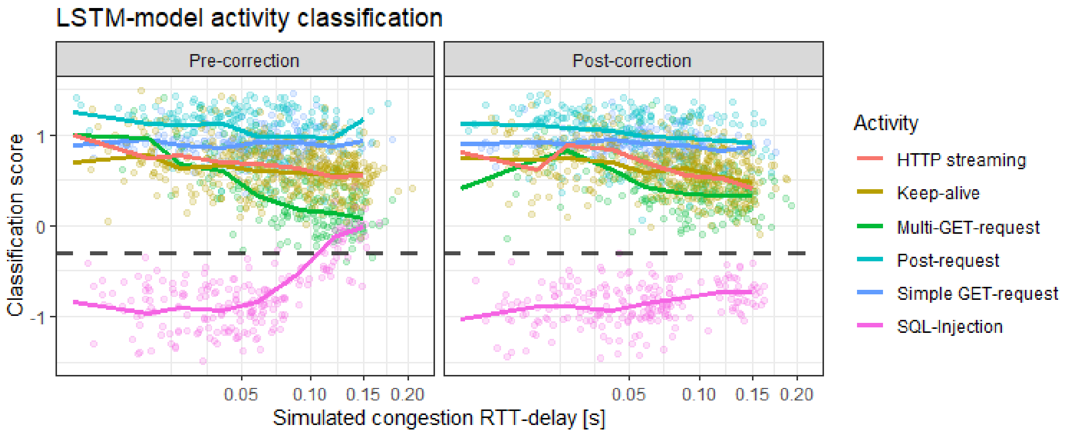


Figure 4.2: Scores for the LSTM-traffic model before and after the model correction.

HTTP-activities generated by DetGen (7.5%) that mirror the characteristics in the training data, as explained in Section 4.5. In total, we use 30,000 connections for training and for evaluating the model, or slightly under 2 million packets. The initially trained model performs relatively well, with an *Area under curve* (AUC)-score of **0.981**, or a detection/false positive rate[1] of **96%** and **2.7%**. However, to enable operational deployment the false positive rate would need to be several magnitudes lower [**?**].

Step 2: Now suppose we want to improve these rates to both detect more SQL-injections and retain a lower false positive rate. To start, we explore which type of connections are misclassified most often. We retrieve the classification scores for all connections and measure their linear correlation to the microstructure labels available for the probing data. The highest misclassification ratio was measured for one of the three SQL injection scenarios (19% correlation) and connections with multiple GET-requests (11% correlation). When not distinguishing activities, we measured a high misclassification correlation with simulated packet latency (12%), which we now examine. More details on this exact procedure can be found in (citation currently blinded).

Step 3: Fig. 4.2 depicts classification scores of connections in the probing data in dependence of the emulated network latency. The left panel depicts the scores for the initially trained model, while the right panel depicts scores after the model correction that we introduce further down. The left panel shows that classification scores are well separated for lower congestion, but increased latency in a connection leads to a narrowing of the classification scores, especially for SQL-injection traffic. Since there are no classification scores that reach far in the opposing area, we conclude that congestion simply makes the model lose predictive certainty. Increased latency can both increase variation in observed packet interarrival times (IATs), and lead to packet out-of-order arrivals and corresponding retransmission attempts. Both of these factors can decrease the overall sequential coherence for the model, i.e. that the LSTM-model loses context too quickly either due to increased IAT variation or during retransmission sequences.

[1] tuned for the geometric mean

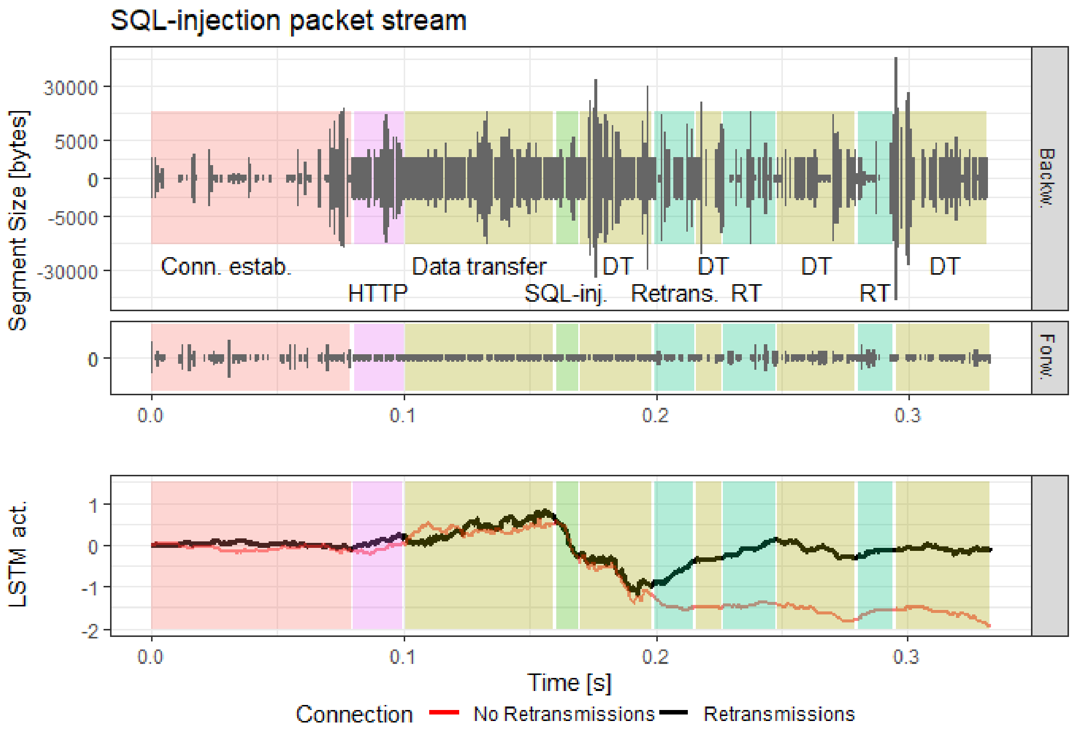


Figure 4.3: LSTM-output activation in dependence of connection phases.

Step 4: We use DetGen to generate two similar connections, where one connection is subject to moderate packet latency and corresponding reordering while the other is not. DetGen's ability to shape traffic in a controlled and deterministic manner allows us to examine the effect of retransmission sequences on the model output and isolate it from other potential influence factors. Fig. 4.3 depicts the evolution of the LSTM-output layer activation in dependence of difference connection phases for the connection subject to retransmissions. Depicted are packet segment streams and their respective sizes in the forward and backward direction, with different phases in the connection coloured and labelled. Below is the LSTM-output activation while processing the packet streams. The red line shows the output for the connection without retransmissions[2] as a comparison. Initially the model begins to view the connection as benign when processing regular traffic, until the SQL-injection is performed. The model then quickly adjusts and provides a malicious classification after processing the injection phase and the subsequent data transfer, just as it is supposed to.

The correct output activation is however quickly depleted once the model processes a retransmission phase and is afterwards not able to relate the still ongoing data transfer to the injection phase and return to the correct output activation. When we compare this to the connection without retransmissions, depicted as the red line in Fig. 4.3, we do not encounter this depletion effect. Instead, the negative activation persists after the injection phase.

Step 5: Based on this analysis, we try to correct the existing model with a simple fix by excluding retransmission sequences at the pre-processing stage. This leads to significantly better classification results during network latency, as visible in the right panel of Fig. 4.2. SQL-injection scores are now far-less

[2] scaled temporally to the same connection phases

affected by congestion while scores for benign traffic are also less affected, albeit to a smaller degree. The overall AUC-score for the model improves to **0.997** while tuned detection rates improved to **99.1%** and false positives to **0.345%**, a five-fold improvement from the previous false positive rate of 2.7%.

4.4 Refining the notion of benign traffic for anomaly detection

Next, we show how ground-truth traffic information can help produce more coherent clusters and thus refine the benign traffic model in anomaly-detection. In particular, we will examine a simplified version of *Kitsune* [?], a recent deep learning anomaly-detection model based on stacked autoencoders. *Kitsune's* AUC-scores surpassed those of other state-of-the-art methods for a variety of attacks, including various types of Botnet traffic and *man-in-the-middle* attacks.

The model takes connection packet streams as input, which are pushed through an artificial information bottleneck before reconstruction, which forces the model to learn and compress reoccurring traffic structures. The compressed connection representation is essentially a positional projection into a lower-dimensional vector space, where spatial boundaries around benign traffic can be drawn. For demonstration purposes, we use a widely-used clustering approach for anomaly-detection rather than *Kitsune's* more complex ensemble method. Here, anomalous outliers are detected using the Mahalanobis-distance of a projected connection from identified cluster centers. Benign traffic should ideally be distributed evenly around the cluster centres to allow a tight borders and good separation from actual abnormal behaviour.

Unstructured datasets such as the CAIDA traffic traces assumably contain too much abnormal behaviour to train an anomaly-detection model, which is why we train the model on benign traffic from the CICIDS-17 [?] intrusion detection dataset (80%). Again, we add 20% probing traffic consists of HTTP, FTP, SSH, and SMTP communication, using a wide spectrum of settings for examination purposes. Attack data for the evaluation was again provided through the CICIDS-17 dataset, and includes access attacks such as SQL-injections or Brute-Forcing, as well as Mirai botnet traffic. We train the model with in total 150,000 connections.

4.4.1 Projection coherency evaluation

Like many approaches that generate representations of benign traffic for anomaly detection, *Kitsune* projects traffic events into a vector-space where traffic clusters and similarities become more apparent. In order for the projection to accurately capture important traffic structures, this projection should be consistent, i.e. traffic events with similar origins and characteristics should be projected to similar positions rather than be dispersed throughout the vector space [?].

To verify the models projection consistency, we generate traffic from near-identical conditions to provide certainty on the expected traffic similarities. We generate a small dataset that consists of HTTP-requests, file-synchronisation, and Botnet communication. For each of the three traffic types we fix four settings

Label	**HTTP**	**File-Sync**	**Mirai-C&C**
1	Get-req. NGINX, low lat.	Two hosts, low lat.	Command 1, low lat.
Results:	0.14 , 0.45	0.19 , 0.27	0.03 , 0.06
2	Multi-req. NGINX, low lat.	Four hosts, low lat.	Command 2, low lat.
Results:	0.32 , 0.45	0.15 , 0.33	0.03 , 0.04
3	Post-req. Apache, high lat.	Two hosts, high lat.	Command 3, high lat.
Results:	0.17 , 0.28	0.16 , 0.28	0.02 , 0.04
4	Multi-req. Apache, high lat.	Four hosts, high lat.	Command 4, high lat.
Results:	0.53 , 2.51	0.71 , 1.31	0.03 , 0.05

Table 4.1: Outline of the traffic settings for examining projection consistency. The numbers below each setting describe the measured Mahalanobis-distances (blue:average, red:maximal) for the corresponding projections.

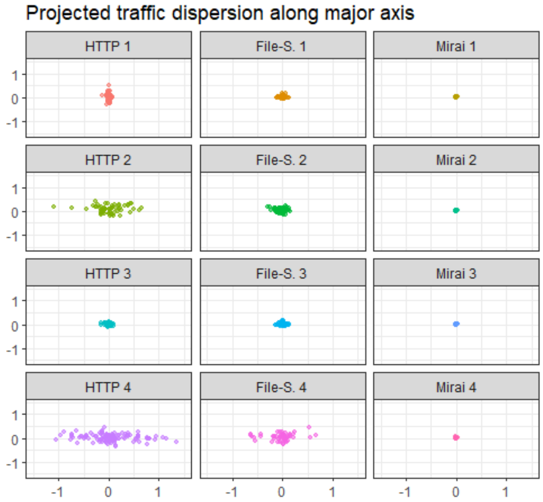


Figure 4.4: Dispersion of projected traffic samples from each setting, plotted along the two most dispersed axes.

that vary in the performed activity and network latency, with the traffic shaping described in Section **??** being held constant within each setting except for small variations in the transmitted message or file. Table 4.1 summarises the traffic for each setting.

We verify if traffic samples within each group are projected to similar areas by measuring the average and maximum Mahalanobis-distance to quantify the overall dispersion of the samples. The results are displayed in Table 4.1 and depicted in Fig. 4.4. The first thing to notice is that the model projects samples from each group within the same cluster, thus confirming the capture of a coarse traffic structure. When looking at the traffic dispersion and the corresponding Mahalanobis-distance measurements, we notice that the *multi-request HTTP* traffic as well as the *file-synchronisation* between mutliple computers is much further dispersed than in the other settings, especially when exposed to more latency. We also find that the corresponding dimension, x_3, with the most projected dispersion seems to be the same for each of the four settings. This suggests that the cause for the dispersion is the same for the different traffic types.

We now focus on the influence of input features on the projected positions exclusively in the x_3-direction. Here, we can again perform a simple correlation analysis between different the input feature values and the corresponding x_3-value. We observe that the arrival time of packet bears the most correlation (5.4%) for the selected settings. We also see that this influence is concentrated primarily on connections that are opened shortly after a previous connection, with the temporal separation between these two connections apparently being the primary cause for the spread on the x_3-axis. The connection interarrival times are naturally an important feature for *Kitsune* to detect attacks such as *Man-in-the-Middle*, which could explain the weight this feature plays in the projection process.

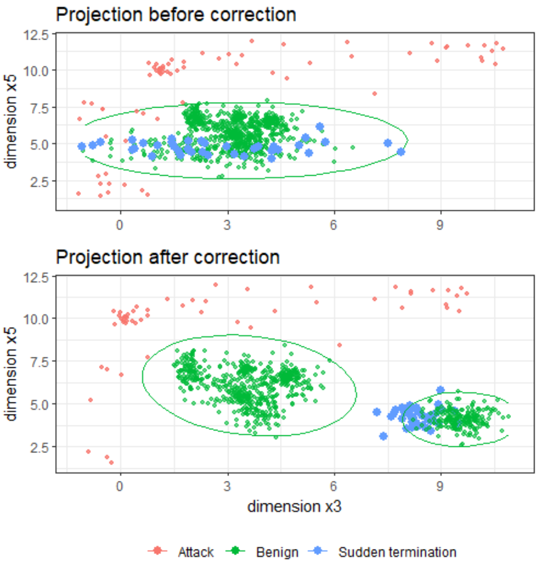


Figure 4.5: Scores for the LSTM-traffic classification model in dependence of simulated network congestion, along with the classification threshold

4.4.2 Investigating individual cluster incoherences

When examining false-positive and corresponding anomaly scores, we noticed that the model often classifies Brute-Force Web attacks as benign and some HTTP-traffic as anomalous. When examining the projected location of the corresponding connections, we see that most of this HTTP-traffic as well as the Brute-Force attack traffic lie near a particular cluster, depicted in Fig. 4.5. A significant portion of traffic in that cluster seems to be spread significantly more across the cluster axis than the rest of the traffic in that cluster, leading to an inflated radius that partially encompasses Brute-Force traffic.

When cross-examining the traffic in this cluster with the probing data, we see that HTTP-traffic with the label ”Sudden termination” are distributed across the cluster axis in a similar fashion, also depicted in Fig. 4.5, suggesting the conclusion that this type of traffic causes the inflated cluster radius. DetGen generates traffic with the label ”Sudden termination” as half-open connections which were dropped by the server due to network failure. One defining characteristic of such connections are that they are not closed with a termination handshake using FIN-flags. To better capture this defining characteristics in the modelling process, we included an additional feature attached to the end of a packet sequence that indicates a proper termination with FIN-flags in the modelling process. The newly trained model now projects ”Sudden termination” connections into a different cluster, which leads to a far better cluster coherence. The detection rate on Brute-Force attack traffic could thus be improved from **89.7%** to **94.1%**.

4.5 Reconstructing an IDS-dataset for efficient probing

Moving towards a more general dataset constructed to apply this probing methodology, we constructed *DetGen-IDS*. This dataset is suitable to quickly probe ML-model behaviour that were trained on the CICIDS-17 dataset [**?**]. The dataset mirrors properties of the CICIDS-17 data to allow pre-trained models to be probed without retraining. The *DetGen-IDS* data therefore serves as complementary probing data that provides microstructure labels and a sufficient and controlled diversity of several traffic characteristics that is not found in the CICIDS-17 data.

We focus on mirroring the following properties from the CICIDS-17 data:

1. **Application layer protocols (ALP)**

2. **ALP implementations**

3. **Typical data volume for specific ALPs**

4. **Conducted attack types**

Extracting more information on characteristics such as conducted activities of current NID-datasets is difficult for the reasons explained in 4.2.2. However, our examination shows that aligning these high-level features with the original training data helps to significantly reduce the validation error of a model on the probing data.

We then took the following steps to extract the necessary information from the CICIDS-17 data and implement the traffic-generation process accordingly:

1. The primary ALPs in the dataset can be identified using their corresponding network ports. We ordered connections by the frequency of their respective port, and excluded connections that do not transmit more than 15 packets per connection as these do not provide enough structure to create probing data from it. This leaves us with the ALPs *HTTP/SSL, SMTP, FTP, SSH, SQL, SMB, and NTP*. We had already implemented traffic scenarios for each of them except SMB and LDAP, which we then added to the catalogue described in Section 7.4.1. Table 4.2 displays the frequency of the most common ALPs in the CICIDS-17 along with their average size and packet number per connection and how we adopted them in the DetGen-IDS data.

2. Most of the used ALP implementations, such as *Apache* and *Ubuntu Webserver* for HTTP, could be gathered from the description of the CICIDS-17 dataset. When this was not the case, it is mostly possible to gather this information by inspecting a few negotiation packets for the corresponding ALP with Wireshark to identify the TLS version or the *OpenSSH*-client. The correct ALP implementation can then be included in the traffic generation process by simply identifying and including a Docker-container that matches the requirements, which is explained more in Section 7.4.2.

3. Since the total size of a connection is one of the most significant features for its classification, we restrict connections in the DetGen-IDS data to cover the same range as their counterparts in the CICIDS-17 data. For this, we extracted the maximum and minimum connection size for each ALP in the benign data and

		CICIDS-17			DetGen-IDS	
ALP	Port	Av. Conn. Size	Av. Packets /Conn.	# Packets	# Packets	# Activities
HTTP	80	131626.4	120.4	26631853	724032	7
HTTPS	443	24637.5	36.7	18531661	432104	7
DNS	53	286.2	3.6	3515510	-	-
SSH	22	4699.6	40.9	430380	379421	13
LDAP	389	5429.2	22.3	133471	94587	3
FTP	21	311.3	41.7	121472	183587	9
NetBIOS	137	773.6	14.3	111341	-	-
SMB	445	12941.5	61.9	88175	47945	3
NTP	123	157.0	3.2	73057	1243	1
SMTP	465	2663.5	21.5	77650	104967	3
Kerberos	88	2687.7	6.9	38262	-	-
mDNS	5353	3685.5	35.5	24592	-	-

Table 4.2: Common ALPs in CICIDS-17 data

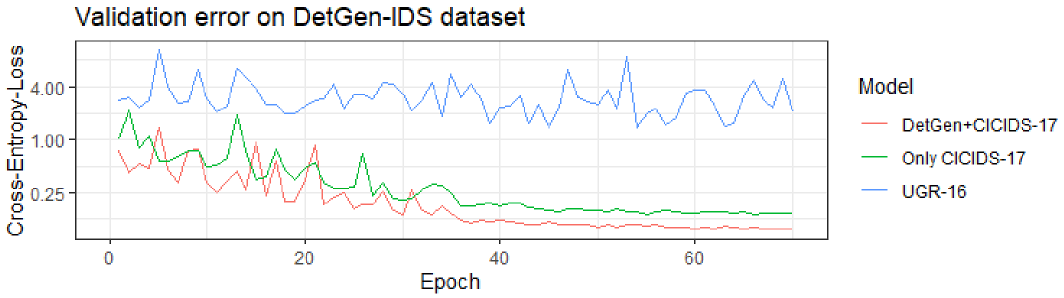


Figure 4.6: Validation errors of LSTM-model [?] on DetGen-IDS data.

use it as a cut-off to remove all connections from the DetGen-IDS data that do not meet this requirement.

4. Included attacks are well documented in the CICIDS-17 description. These include *SQL-injections, SSH-brute-force, XSS, Botnet, Heartbleed, GoldenEye, and SlowLoris*. We aim to cover as many of these attack types in the DetGen-IDS data as well as adding them to the overall DetGen-attack-catalogue. We were not able to cover all attacks though as DetGen either did not provide the necessary network topology to conduct the attack, such as for port-scanning, or the attack types are not implemented in the catalogue of scenarios yet.

In addition to the `pcap`-files, we used the *CICFlowMeter* to generate the same 83 flow-features as included in the CICIDs-17 data. Table 4.2 displays the content and statistics of the DetGen-IDS data.

In Fig. 4.6, we compare the validation error of a recent LSTM-model for network intrusion detection by Clausen et al. [?] on the DetGen-IDS data to demonstrate that a model trained on the CICIDS-17 data is able to perform well without retraining. We distinguish models when trained exclusively on the CICIDS data (green), and when also trained on the probing data (red). Even though the validation error is slightly higher when only trained on the CICIDS data, the difference is almost negligible compared to the error resulting from a model trained on a completely different dataset (UGR-16 [?], blue). This does not fully prove that every model is able to transfer observed structures between the two datasets, but it gives an indicator that they mirror characteristics.

4.6 Traffic microstructures and their influence factors

The biggest and most obvious influence on traffic microstructures is the choice of the application layer protocols. For this reason, the range of protocols is often used as a measure for the diversity of a dataset. However, while the attention to microstructures in current NID-datasets stops here, computer communication involves a myriad of other different computational aspects that shape observable traffic microstructures. Here, we highlight and quantify the most dominant ones, which will act as a justification for the design choices we outline in Section 4.7.4. We look at both findings from previous work as well as our own experimental results.

1. Performed task and application. The conducted computational task as well as the corresponding application ultimately drives the communication between computers, and thus hugely influences characteristics such as the direction of data transfer, the duration and packet rate, as well as the number of connections established. These features are correspondingly used extensively in application fingerprinting, such as by Yen et al. [?] or Stober et al. [?].

2. Application layer implementations. Different implementations for TLS, HTTP, etc. can yield different computational performance and can perform handshakes differently and differ in multiplexing channel prioritisation, which can significantly impact IAT times and the overall duration of the transfer, as shown in a study by Marx et al. [?] for the QUIC/HTTP3 protocol[3].

3. LAN and WAN congestion. Low available bandwidth, long RTTs, or packet loss can have a significant effect on TCP congestion control mechanisms that influence frame-sizes, IATs, window sizes, and the overall temporal characteristic of the sequence, which in turn can influence detection performance significantly as shown in Section 4.3.

4. Host level load. In a similar manner, other applications exhibiting significant computational load (CPU, memory, I/O) on the host machine can affect the processing speed of incoming and outgoing traffic, which can again alter IATs and the overall duration of a connection. An example of this is visible in Fig. 4.7, where a FTP-client sends significantly fewer `PUSH`-packets when under heavy computational load. Colours indicate packet flags while the height of the packets indicates their size. This effect is dependent on the application layer protocol, where at a load number of 3.5 we see about 60% less upstream data-packets while the downstream is only reduced by 10%, compared to HTTP where both downstream and upstream packet rates are throttled by about 40%.

5. Caching/Repetition effects. Tools like cookies, website caching, DNS caching, known hosts in SSH, etc. remove one or more information retrieval requests from the communication, which can lead to altered packet sequences

[3] Fig. 2 in [?] illustrates these differences in a nice way

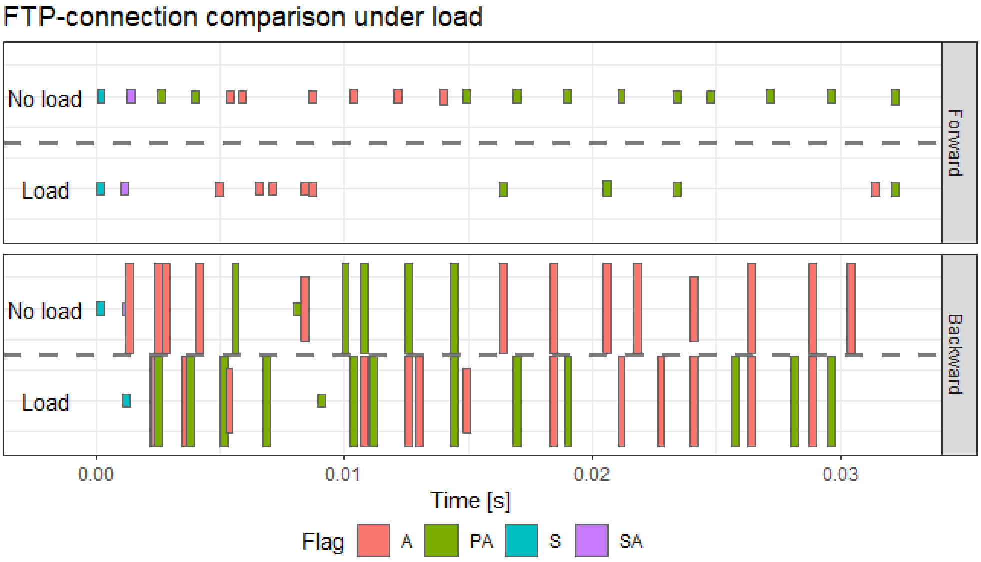


Figure 4.7: Packet-sequence similarity comparison under different load.

and less connections being established. For caching, this can result in less than 10% of packets being transferred, as shown by Fricker et al. [?].

6. User and background activities. The choice and usage frequency of an application and task by a user, sometimes called *Pattern-of-Life*, governs the larger-scale temporal characteristic of a traffic capture, but also influences the rate and type of connections observed in a particular time-window [?]. The mixing of different activities in a particular time-window can severely impact detection results of recent sequential connection-models, such as by Radford et al. [?] or by Clausen et al. [?]. To quantify this effect, we look at FTP-traffic in the CICIDS-17 dataset. As explained in Section 4.2.2, the FTP-traffic overwhelmingly corresponds to the exact same isolated task, and should therefore spawn the same number of connections in a particular time window. However, we observe additional connections from other activities within a 5-second window for 68% of all FTP-connections, such as depicted in Table 4.3, which contains FTP-, HTTPS- and DNS-, as well as additional unknown activity.

Time	Source-IP	Destination-IP	**Dest. Port**
13:45:56.8	192.168.10.9	192.168.10.50	**21**
13:45:56.9	192.168.10.9	192.168.10.50	**10602**
13:45:57.5	192.168.10.9	69.168.97.166	**443**
13:45:59.1	192.168.10.9	192.168.10.3	**53**
13:46:00.1	192.168.10.9	205.174.165.73	**8080**

Table 4.3: 5-second window for host 192.168.10.9 in the CICIDS-17 dataset.

Other prominent factors that we found less in effect on traffic microstructures include:

7. Networking stack load. TCP or IP queue filling of the kernel networking stack can increase packet waiting times and therefore IATs of the traffic trace, as shown by [?]. In practice, this effect seems to be constrained to large WAN-servers and routers. When varying the stack load in otherwise constant settings on an Ubuntu-host, we did not find any notable effect on packet sequences when comparing the corresponding traffic with a set of three similarity metrics. More details on this setting and the metrics can be found in Section 4.8.

8. Network configurations. Network settings such as the MTU or the enabling of TCP Segment Reassembly Offloading have effects on the captured packet sizes, and have been exploited in IP fragmentation attacks. However, these settings have been standardised for most networks, as documented in the CAIDA traffic traces [?].

We designed DetGen to control and monitor factors 1-6 to let researchers explore their impact on their traffic models, while omitting factors 7 and 8 for the stated reasons.

4.7 DetGen: precisely controlled data generation

4.7.1 Design overview

Detgen is a container-based network traffic generation framework that distinguishes itself by providing precise control over various traffic characteristics and providing extensive ground-truth information about the traffic origin. In contrast to the pool of programs running in a VM-setup, such as used in the generation of the CICIDS-17 and 18 [?], or UGR-16 [?], DetGen separates program executions and traffic capture into distinct containerised environments in order to shield the generated traffic from external influences.Traffic is generated from a set of scripted *scenarios* that define the involved devices and applications and strictly control corresponding influence factors. Fig. 4.8 provides a comparison of the DetGen-setup and traditional VM-based setups and highlights how control and monitoring is exerted.

4.7.2 Containerization and activity isolation

As we will demonstrate in Section 4.8, containers provide significantly more isolation of programs from external effects than regular OS-level execution. This isolation enables us to monitor processes better and create more accurate links between traffic events and individual activities than on a virtual machine were multiple processes run in parallel and generate traffic. The corresponding one-to-one correlation between processes and network traces allows us to capture traffic directly from the process and produce labelled datasets with granular ground truth information.

Additionally, containers are specified in an image-layer, which is unaffected during the container execution. This allows containers to be run repeatedly whilst always starting from an identical state, allowing a certain level of **determinism** and reproducibility in the data generation.

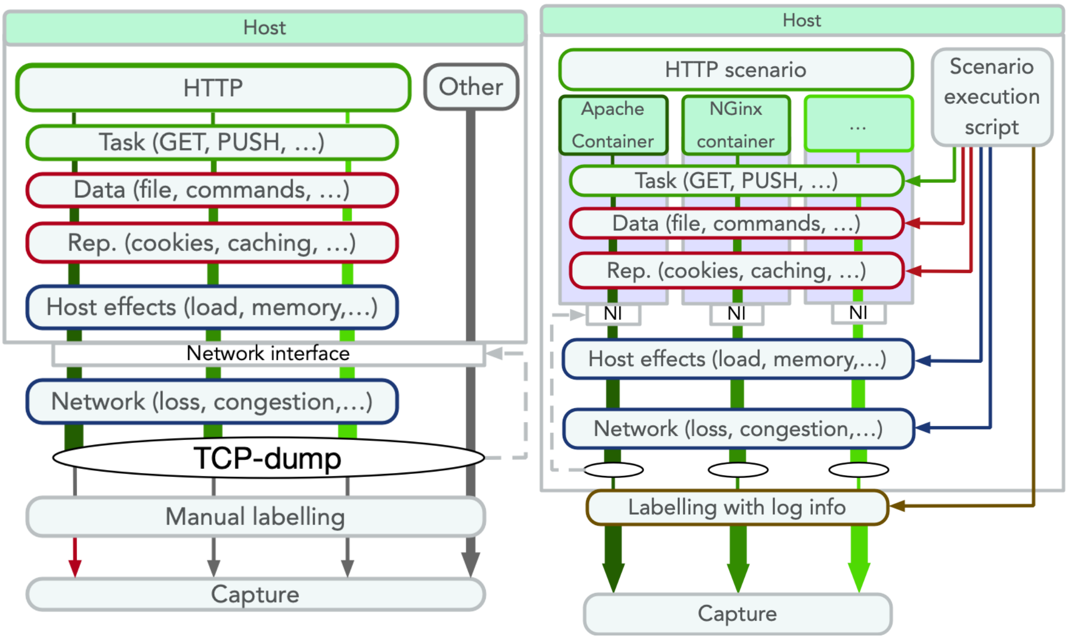

Host
HTTP
Other
Task (GET, PUSH, ...)
Data (file, commands, ...)
Rep. (cookies, caching, ...)
Host effects (load, memory,...)
Network interface
Network (loss, congestion,...)
TCP-dump
Manual labelling
Capture
Host
HTTP scenario
Apache Container
NGinx container
...
Scenario execution script
Task (GET, PUSH, ...)
Data (file, commands, ...)
Rep. (cookies, caching, ...)
NI
NI
NI
Host effects (load, memory,...)
Network (loss, congestion,...)
Labelling with log info
Capture

Figure 4.8: Traditional traffic-generation-setups (left), and DetGen (right).

4.7.3 Activity generation

Scenario.

We define a *scenario* as a composition of containers conducting a specific interaction. Each scenario produces traffic from a setting with two (client/server) or more containers, with traffic being captured from each container's perspective. This constructs network datasets with total interaction capture, as described by Shiravi et al. [?]. Examples may include an FTP interaction, an online login form paired with an SQL database, or a C&C server communicating with an open backdoor. Our framework is modular, so that individual scenarios are configured, stored, and launched independently. We provide a complete list of implemented scenarios in Table 7.1 in the Appendix.

Task.

To provide a finer grain of control over the traffic to be generated, we create a catalogue of different tasks that allow the user to specify the manner in which a scenario should develop. To explore the breadth of the corresponding traffic structures efficiently, we prioritise tasks that cover aspects such as the direction of file transfers (e.g. GET vs POST for HTTP), the amount of data transferred (e.g. HEAD/DELETE vs GET/PUT), or the duration of the interaction (e.g. persistent vs non-persistent tasks) as much as possible. For each task, we furthermore add different failure options for the interaction to not be successful (e.g. wrong password or file directory).

4.7.4 Simulation of external influence

Caching/Cookies/Known server.

Since we always launch containers from the same state, we prevent traffic impact from **repetition effects** such as caching or known hosts. If an application provides caching possibilities, we implement this as an option to be specified before the traffic generation process.

Network effects.

Communication between containers takes place over a virtual bridge network, which provides far higher and more reliable throughput than in real-world networks [?]. To retard and control the network reliability and congestion to a realistic level, we rely on *NetEm*, an enhancement of the Linux traffic control facilities for emulating properties of wide area networks from a selected network interface [?].

We apply NetEm to the network interface of a given container, providing us with the flexibility to set each container's network settings uniquely. In particular, packet delays are drawn from a Paretonormal-distribution while packet loss and corruption are drawn from a binomial distribution, which has been found to emulate real-world settings well [?]. Distribution parameters such as mean or correlation as well as available bandwidth can either be manually specified or drawn randomly before the traffic generation process.

Host load.

We simulate excessive computational load on the host with the tool *stress-ng*, a Linux workload generator. Currently, we only stress the CPU of the host, which is controlled by the number of workers spawned. Future work will also include stressing the memory of a system. We have investigated how stress on the network sockets affects the traffic we capture without any visible effect, which is why we omit this variable here.

4.7.5 Data generation

Execution script.

DetGen generates traffic through executing execution script that are specific to the scenario. The script creates the virtual network and populates it with the corresponding containers. The container network interfaces of the containers are then subjected to the NetEm chosen settings and the host is assigned the respective load before the inputs for the chosen task are prepared and mounted to the containers.

Labelling and traffic separation.

Each container network interface is hooked to a *tcpdump*-container that records the packets that arrive or leave on this interface. Combined with the described process isolation, this setting allows us to exclusively capture traffic that corresponds to the conducted activity and exclude any background events. The

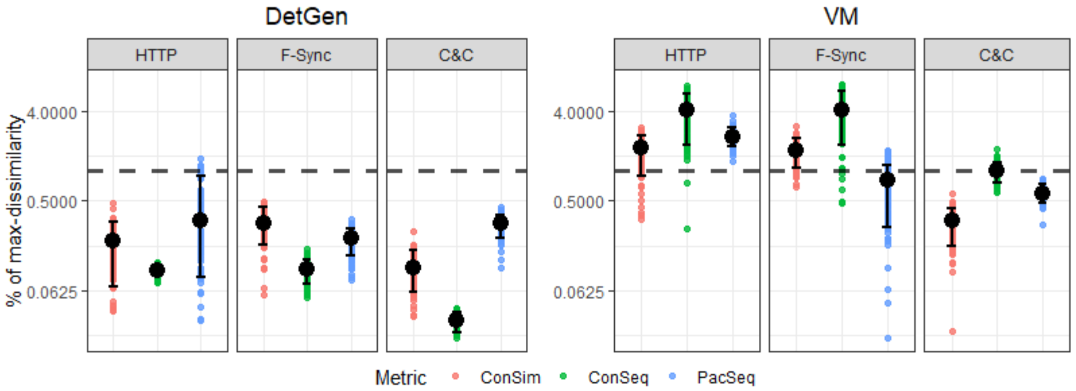


Figure 4.9: Dissimilarity scores for DetGen and a regular VM-setup, on a log-scale.

execution script then stores all parameters (conducted task, mean packet delay, ...) and descriptive values (input file size, communication failure, ...) for the chosen settings.

4.8 Traffic control and generative determinism of DetGen

We now assess the claim that DetGen controls the outlined traffic influence factors sufficiently, and how similar traffic generated with the same settings looks like. We also demonstrate that this level of control is not achievable on regular VM-based NIDS-traffic-generation setup.

To do so, we generate traffic from three traffic types, namely HTTP, file-synchronisation, and botnet-C&C, each in four configurations that varied in terms of conducted activity, data/credentials as well as the applied load and congestion. Within each configuration all controllable factors are held constant to test the experimental determinism and reproducibility of DetGen's generative abilities. As a comparison, we use a regular VM-based setup, were applications are hosted directly on two VMs that communicate over a virtual network bridge that is subject to the same NetEm effects as DetGen, such as depicted in Fig. 4.8. Such a setup is for example used in the generation of the UGR-16 data [?].

To measure how similar two traffic samples are, we devise a set of similarity metrics that measure dissimilarity of overall connection characteristics, connection sequence characteristics, and packet sequence characteristics:

- **Overall connection similarity** We use the 82 flow summary statistics (IAT and packet size, TCP window sizes, flag occurrences, burst and idle periods) provided by CICFlowMeter [?], and measure the cosine similarity between connections, which is also used in general traffic classification [?].

- **Connection sequence similarity** To quantify the similarity of a sequence of connections in a retrieval window, we use the following features to describe the window, as used by Yen et al. [?] for application classification: The number of connections, average and max/min flow duration and size,

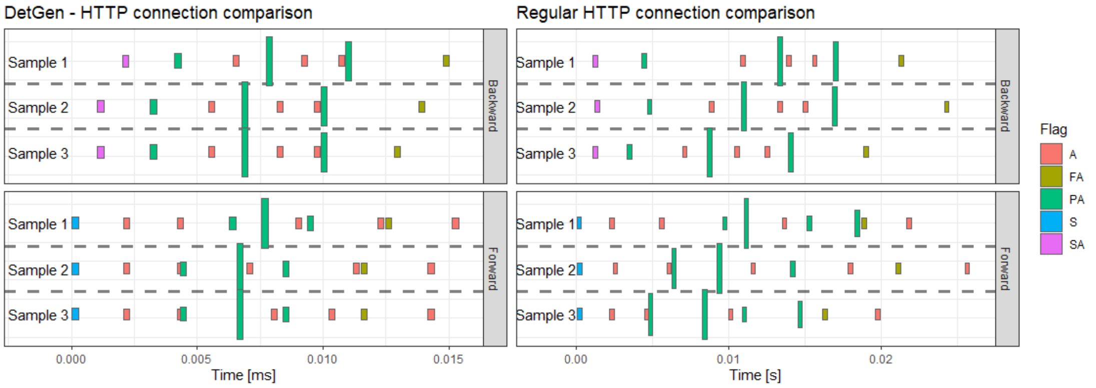


Figure 4.10: Packet-sequence similarity comparison for HTTP-activity for DetGen and a regular setting.

number of distinct IP and ports addresses contacted. We then again measure the cosine similarity based on these features between different windows.

- **Packet sequence similarity** To quantify the similarity of packet sequences in handshakes etc., we use a Markovian probability matrix for packet flags, IATs, sizes, and direction conditional on the previous packet. We do this for sequences of 15 packets and use the average sequence likelihood as this accommodates better for marginal shifts in the sequence.

We normalise all dissimilarity scores by dividing them by the maximum dissimilarity score measured for each traffic type to put the scores into context. For each configuration, we generate 100 traffic samples and apply the described dissimilarity measures to 100 randomly drawn sample pairs. Fig. 4.9 depicts the resulting dissimilarity scores on a log-scale.

The DetGen-scores yield consistently less than 1% of the dissimilarity observed on average for each activity. Scores are especially low when compared to traffic groups collected in the VM setting, which are consistently more dissimilar, in particular for connection-sequence metrics, where the average dissimilarity is more than 30 times higher than for the DetGen setting. Manual inspection of the VM-capture showed that high dissimilarity is caused by additional flow events from background activity (OS and application HTTP, NTP, DNS, device discovery) being present in about 24% of all captures. . While sequential dissimilarity is roughly the same for the DetGen- and the VM-settings, overall connection similarity for the VM-setting sees significantly more spread in the dissimilarity scores when computational load is introduced.

Fig. 4.10 depicts an exemplary comparison between HTTP-samples generated using DetGen versus generation using the VM-setup. Colours indicate packet flags while the height of the packets indicates their size. Even though samples from DetGen are not perfectly similar, packets from the VM-setup are subject to more timing perturbations and reordering as well as containing additional packets. Additionally, the packet sizes vary more in the regular setting.

These results confirm that DetGen exerts a high level of control over traffic shaping factors while providing sufficient determinism to guarantee ground-truth traffic information.

4.9 Conclusions

In this paper, we described and examined a tool for generating traffic with controllable and extensively labelled traffic microstructures with the purpose of probing machine-learning-based traffic models. For this, we demonstrated the impact that probing with carefully crafted traffic microstructures can have for improving a model with a state-of-the-art LSTM-traffic-classifier with a detection rate that improved by more than 3% after understanding how the model processes excessive network congestion.

To verify DetGen's ability to control and monitor traffic microstructures, we performed experiments in which we quantified the experimental determinism of DetGen and compared it to traditional VM-based capture setups. Our similarity metrics indicate that traffic generated by DetGen is on average 10 times, and for connection sequences up to 30 times more consistent.

Alongside this work, we are releasing DetGen-IDS, a substantial dataset suitable for probing models trained on the CICIDS-17 dataset. This data should make it easier for researchers to understand where their model fails and what traffic characteristics are responsible to subsequently improve their design accordingly.

DetGen and the corresponding dataset are openly accessible for researchers on GitHub.

Difficulties and limitations:

While the control of traffic microstructures helps to understand packet- or connection-level models, it does not replicate realistic network-wide temporal structures. Other datasets such as UGR-16 [?] or LANL-15 [?] are currently better suited to examine models of large-scale traffic structures.

While controlling traffic shaping factors artificially helps at identifying the limits and weak points of a model, it can exaggerate some characteristics in unrealistic ways and thus alter the actual detection performance of a model.

The artificial randomisation of traffic shaping factors can currently not completely generate real-world traffic diversity. This problem is however more pronounced in commonly used synthetic datasets such as CICIDS-17, where for example most FTP-transfers consist of a client downloading the same text file.

Discussions about the implications of the model correction proposed in Section 4.3 are above the scope of this paper, and there likely exist more complex and suitable solutions.

Future work:

DetGen is currently only offering insufficient control over underlying **application-layer implementations** such as TLS 1.3 vs 1.2. In theory, it should be unproblematic to provide containers with different implementations, and we are currently investigating how to compile containers in a suitable manner.

We are currently investigating how to better simulate causality in connection spawning and other **medium-term temporal dependencies**, such as by importing externally generated activity timelines from tools such as Doppelganger [?].

A project we are currently working on is to embed traffic scenarios into a larger and more complex **network topology** using MiniNet [**?**].

Chapter 5

Flow model

5.1 CBAM: A contextual model for network anomaly detection

Anomaly-based intrusion detection methods are aimed to combat the increasing rate of zero-day attacks [?], but their success is currently restricted to the detection of high-volume attacks using aggregated traffic features. Recent evaluations show that the current anomaly-based network intrusion detection methods fail to detect remote access attacks reliably [?]. These are smaller in volume and often only stand out when compared to their surroundings. Currently, anomaly methods try to detect access attack events mainly as point anomalies and neglect the context they appear in. We present and examine CBAM, a contextual bidirectional anomaly model based on deep LSTM-networks that is designed specifically to detect such attacks as contextual network anomalies. The model efficiently learns short-term sequential patterns in network flows as conditional event probabilities. Access attacks frequently break these patterns when exploiting vulnerabilities, and can thus be detected as contextual anomalies. We evaluated our CBAM on an assembly of three datasets that provide both representative network access attacks, real-life traffic over a long timespan, and traffic from a real-world red-team attack. We contend that this assembly is closer to a potential deployment environment than current NIDS benchmark datasets. We show that by building a deep model, we are able to reduce the false positive rate to 0.16% while detecting effectively, which is significantly lower than the operational range of other methods. We furthermore demonstrate that short-term flow structures remain stable over long periods of time, making CBAM robust against concept drift.

5.2 Introduction

Remote access attacks are used to gain control or access information on remote devices by exploiting vulnerabilities in network services, and are involved in many of today's data breaches [?]. A recent survey [?] showed that these attacks are detected at significantly lower rates than more high-volume probing or DoS attacks. We have recently presented **CBAM**, a short-term **c**ontextual **b**idirectional **a**nomaly **m**odel of network flows at the MLN 2020 conference [?], which improves detection rates of remote access attacks significantly. The underlying idea of CBAM is to capture probability distributions over sequences of network flows that quantify their overall likelihood, much like a language model. CBAM is based on deep bidirectional LSTM networks.

We have evaluated CBAM carefully on two modern network intrusion detection datasets. Furthermore, we reimplemented and evaluated three state-of-the-art methods on these datasets and compared their performance against ours. By carefully selecting input parameters based on their sequential interdependence as well as increasing model complexity in terms of depth and efficient input embedding compared to preceding models, we are able to detect remote access attacks at a false positive rate of 0.16%, a rate at which none of the comparison models are able to detect any attacks reliably. Here, we present results from an additional real-world dataset, LANL-15, that display how and why CBAM can detect the real-world attacks contained in the dataset.

We also discuss specific design choices and how they enable effective modelling of specific traffic characteristics to boost performance. Recently, deep learning models such as LSTMs have been a popular tool in network intrusion detection [?, ?, ?]. However, the evaluation of these models is generally agnostic to particular characteristics of the modelled traffic and fails to explain where and why the corresponding model fails to classify traffic properly. We have recently demonstrated how a detailed examination of these failings of two state-of-the-art NID-models enables specific improvements in the model design to boost detection performances [?]. The additional results presented here aim to do the same for our short-term CBAM model and provide a validation for the undertaken design steps.

5.2.1 Contribution

This paper extends our work "Better Anomaly Detection for Access Attacks Using Deep Bidirectional LSTMs", presented at the 3rd International Conference on Machine Learning for Networking (MLN'2020) [?], and provides additional traffic examinations and corresponding performance results to demonstrate the design process. In particular, we present the following contributions:

- We provide extensive examination how common access attacks perturb short-term contextual sequence structures, and can thus be detected as anomalies by our presented model.

- We examine in detail how adding bidirectional LSTM-layers, increasing network depth, and adding a separate size vocabulary helps CBAM predict flows better for specific traffic types to reduce false-positives.

- We provide additional performance results on the LANL-15 real-world dataset to demonstrate that CBAM is capable of detecting real-world attacks in real-world traffic.

- We examine in more detail how short-term flow sequence structures remain stable over long time periods, what the most common source for false-positives are, and how the size of the training data affects the models ability to reliably recognise benign traffic.

5.2.2 Outline

The remainder of the paper is organised as follows: Section 5.3 provides a motivation for short-term contextual models and their benefits for the detection of access attacks. Section 7.4 explains the methodology and architecture of CBAM as well as the data preprocessing. Section 5.5 describes the problems with network traffic datasets which previous methods were evaluated on, and explains the advantages of our selection of datasets. We also describe how and why traffic from particular hosts is selected, and how training and test data are constructed. Section 5.6 discusses our detection rates on attack traffic in the CICIDS-17 and LANL-15 data, and examines how and why CBAM is able to identify these attacks. Section 5.7 discusses the false positive rate on benign traffic, and examines long-term stability of flow structures and corresponding model performance. It also provides details on the score distributions and the type of traffic that is both predicted accurately and inaccurately, as well as the influence of the training data size on these predictions. Section 5.8 discusses the reason and measured benefit of specific design steps that increase model complexity. Section 5.11 concludes our results.

5.3 Overview

Src	Dst	DPort	bytes	# packets
A	B	80	247956	315
A	B	80	7544	13
A	B	80	328	6
A	B	80	2601	10
A	B	80	328	6
A	B	80	328	6
A	B	80	380	7
A	B	80	328	6
			⋮	

(a) XSS-attack, A=192.168.10.50, B= 172.16.0.1

Src	Dst	DPort	bytes	# packets
D	C	N33	600	5
C	D	445	77934	1482
D	C	N33	600	5
C	D	445	5202	10

(b) Benign SMB, C=C6267, D=C754

Src	Dst	DPort	bytes	# packets
C	D	445	4106275	2830
C	D	445	358305611	242847

(c) *Pass-the-hash* attack via SMB

Table 5.1: The left side depicts a flow sequence from an XSS-attack.The right side depicts a benign SMB-sequence (top), and a sequence from a *Pass-the-hash* attack via the same SMB service.

In verbal or written speech, we expect the words “I will arrive by . . . ” to be followed by a word from a smaller set such as “car” or “bike” or “5pm”. Similarly, on an average machine we may expect DNS lookups to be followed by outgoing HTTP/HTTPS connections. These short-term structures in network traffic are a reflection of the computational order of information exchange. Attacks that exploit vulnerabilities in network communication protocols often achieve their target by deviating from the regular computational exchange of a service, which should be reflected in the generated network pattern.

Table 1(a) depicts a flow sequence from an XSS-attack. Initial larger flows are followed by a long sequence of very small flows which are likely generated by

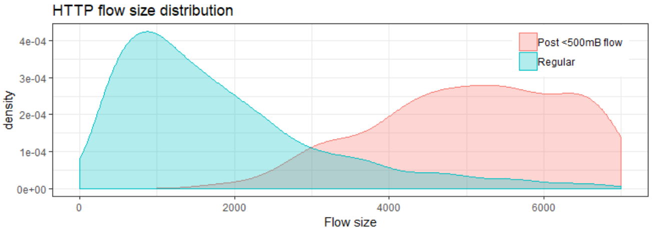


Figure 5.1: HTTP flow size distribution overall, and if preceded by an HTTP flow smaller than 500 bytes.

the embedded attack script trying to download multiple inaccessible locations. Flows of this size are normally immediately followed by larger flows, as depicted in Fig. 5.1, which makes the repeated occurrence of small HTTP flows in this sequence very unusual.

Table 1(b) depicts a regular SMB service sequence while Table 1(c) depicts a *Pass-the-hash* attack via the same SMB service. As shown, the flows to port N33 necessary to trigger the communication on the SMB port are missing while the second flow is significantly larger than any regular SMB flows due to it being misused for exfiltration purposes.

The underlying idea of CBAM is to predict probabilities of connections in a host's traffic stream conditional on adjacent connections. The probabilities are assigned based on the connection's protocol, network port, direction, and size, and the model is trained to maximise the overall predicted probabilities.

To assign probabilities, we map each connection event to two discrete sets of states, called vocabularies, according to the protocol, the network port, and the direction of the connection for the first, and according to number of transmitted bytes for the second. The size of the vocabulary is chosen large enough to capture meaningful structures without capturing rare events that can deteriorate prediction quality. We feed these vocabularies into a deep bidirectional LSTM (long short-term memory) network that takes bivariate sequences of mapped events as input to efficiently capture the conditional probabilities for each event.

CBAM acts as an anomaly-detection model that learns short-term structures in benign traffic and identifies malicious sequences as deviations from these structures. By predicting probabilities of flows in benign flow sequences, CBAM is trained in a self-supervised way on strictly benign traffic. In contrast to classification-based training, CBAM does not require labelled attack traffic in the training data and is thus not affected by typical class imbalances in network intrusion datasets.

5.4 Design

5.4.1 Session construction

A network flow (often referred to as "NetFlow") is a summary of a connection between two computers and contains a timestamp, the used IP protocol, the source and destination IP address and network port, and a choice of summary values. The raw input data, in the form of network flows, contains unordered traffic from and to all hosts in the network. To order the raw network flows,

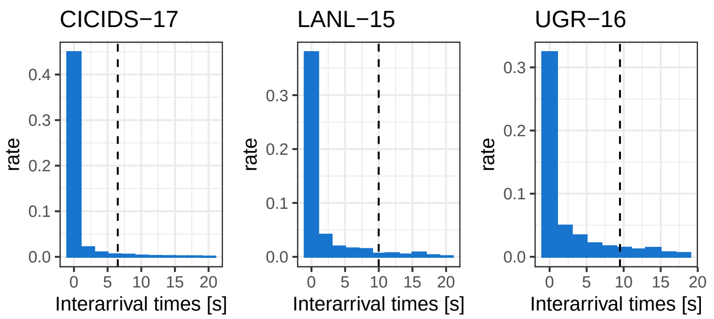


Figure 5.2: Flow interarrival distributions for selected hosts in the the CICIDS-17, the LANL-15, and the UGR-16 data, with 90 percent quantile lines.

we first gather all outgoing and incoming flows for each of the hosts selected for examination according to their IP address.

The traffic a host generates is often seen as a series of *session*, which are intervals of time during which the host is engaging in the same, continued, activity [?]. In our context, flows that occur during the same session can be seen as having strong short-term dependencies. We therefore group flows going from or to the same host to sessions using an established statistical approach [?]:

If a network flow starts less than α seconds after the previous flow for that host, then it belongs to the same session; otherwise a new session is started. If a session exceeds β events, a new session is started.

We chose the number of $\alpha = 8$ seconds as we have found that on average around 90% of flows on a host start less than 8 seconds after the previous flow, a suitable threshold to create cohesive sessions according to Rubin-Delanchy et al. [?]. We introduced the β parameter in order to break up long sessions that potentially contain a small amount of malicious flows, and estimated $\beta = 25$ to be a suitable parameter. Detection rates do not seem to be very sensitive to the exact choice of β though.

A perfect session grouping would require (unavailable) information from the top layers of the network stack. We therefore use our session definition as a first approximation which we found to be useful enough for this experiment. We will discuss this issue further in Section 5.8.2 and Section 5.10.

The interarrival time distribution for selected hosts in the used datasets, described in Section 5.5.1, along with 90% quantile lines is depicted in Fig. 5.2.

5.4.2 Contextual modelling

Each session is now a sequence of flows that are assumed to be interdependent. We observed in an initial traffic analysis that the protocol, port, and direction of a flow as well as its size are highly dependent on the surrounding flows, which motivates their use in the modelling process. We treat flows as symbolic events that can take different states, much like words in a language model. The state of a flow is defined as the tuple consisting of the protocol, network port, and the direction of the flow. We consider only the server port numbers, which indicate the used service, in the state-building process. We introduce the following notation:

M:	number of states
C:	number of host groups
S:	number of size groups
N^i_{embed}:	embedding dimension
N^i_{hidden}:	LSTM layers dimension
N_j:	the length of session j
$x^{i,j} \in \{1, \ldots, M\}$:	the state of flow i in session j
$c^j \in \{1, \ldots, C\}$:	the host group
$s^{i,j} \in \{1, \ldots, S\}$:	size group of flow i in session j
$p_x^{i,j,k} = P(x^{i,j} = k \vert j)$	the predicted probability of $x^{i,j} = k$ conditional on the other flows in session j
$p_s^{i,j,l} = P(s^{i,j} = l \vert j)$	the predicted probability of $s^{i,j} = l$ conditional on the other flows in session j

The collection of all states is called a *vocabulary*. For prediction, the total size of a vocabulary directly correlates with the number of parameters needed to be inferred in an LSTM network, thus influencing the time and data volume needed for training. Too large vocabularies also lead to decreased predictive performance by including rare events that are hard to predict [**?**]. We therefore bound the total number of states and only distinguish between the $M - 2$ tuples of protocol, port, and direction most commonly seen on a machine, with less popular combinations being grouped as "Other". Furthermore, the end of a session is treated as an additional artificial event with its own state. The total vocabulary size is then given by M.

Our experimentation has shown that detection rates improve when including the size as an additional variable, as we discuss in Section 5.6.3. Rather than making a point estimate of the size, we want to produce a probability distribution for different size intervals. This provides better accuracy for situations in which both small and large flows have a similar occurrence likelihood. We group flows into S different size quantile intervals, with the set of all size intervals forming a third vocabulary. The $S-1$ boundaries that separate the size intervals correspond to $S - 1$ equidistant quantiles of the size distribution in the training data.

Hosts are grouped according to their functionality (Windows, Ubuntu, servers, etc.) a distinction that can easily be performed using signals in the traffic. The group is provided to the model as an additional input parameter c^j and forms a third vocabulary.

5.4.3 Architecture selection

We now represent each session as a set of two symbolic sequences that contain between three and 27 items, in order to capture their contextual structure for the reasons described in Section 5.3. A number of techniques exist to describe such sequences, such as *Markov-models and Hidden-Markov-models, Finite-State-Automata,* or *N-Gram* models. However, the success of recurrent neural networks

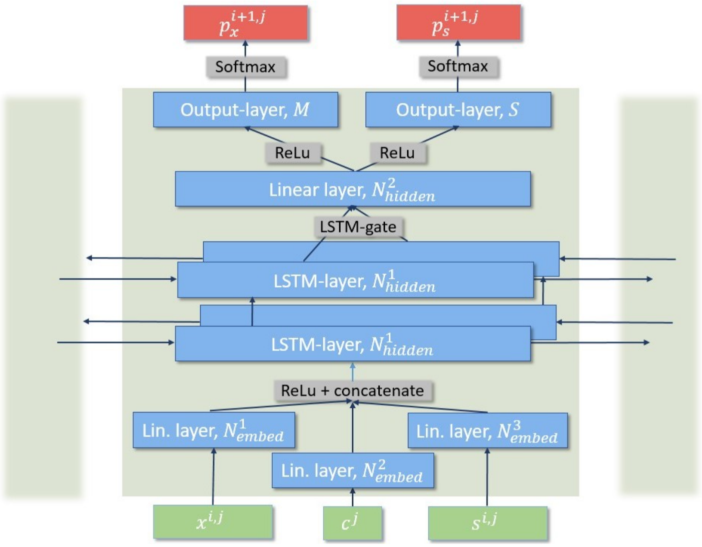

$p_x^{i+1,j}$
$p_s^{i+1,j}$
Softmax
Softmax
Output-layer, M
Output-layer, S
ReLu
ReLu
Linear layer, N_{hidden}^2
LSTM-gate
LSTM-layer, N_{hidden}^1
LSTM-layer, N_{hidden}^1
ReLu + concatenate
Lin. layer, N_{embed}^1
Lin. layer, N_{embed}^3
Lin. layer, N_{embed}^2
$x^{i,j}$
c^j
$s^{i,j}$

Figure 5.3: Architecture of the trained bidirectional LSTM network.

in similar applications of natural language processing over these methods suggests they would be the most appropriate architecture to capture contextual relationships between flows. In Section 5.8.3, we compare the performance of CBAM to both Markov-based models and Finite-State-Automata. Even though convolutional neural networks and feed-forward networks can be more suitable choices for specific sequential problems with tabular or regression characteristics, recurrent neural networks such as LSTMs or GRUs normally outperform them for short tokenised sequences [?]. Both LSTMs and GRUs perform similarly well and generally outperform simple RNNs.

5.4.4 Trained architecture

We use a deep bidirectional LSTM network which processes a sequence in both forward and reverse direction to predict the state and size group of individual flows. The architecture of the network we trained is depicted in Fig. 5.3. The increased model complexity we present has not been explored in previous LSTM applications to network intrusion detection, and enables us to boost detection rates while lowering false positive rates, which we demonstrate in Section 5.8.

Embedding

First, each of the three vectors is fed through an embedding layer, which assigns them a vector of size N^i_{embed}, $i \in \{1, 2, 3\}$. This embedding allows the network to project the data into a space with easier temporal dynamics. This step significantly extends existing designs of LSTM models for anomaly detection and allows us to project multiple input vocabularies simultaneously without a large increase in the model size. By treating the state, the size group, and the host group as separate dictionaries, we avoid the creation of one large vocabulary of size $M \times C \times S$, which makes training faster and avoids the creation of rare states [?].

LSTM-layer

In the second step, the vectors are concatenated and fed to a stacked bidirectional LSTM layer with N^1_{hidden} hidden cells. This layer is responsible for the transport of sequential information in both directions. The usage of bidirectional LSTM layers compared to unidirectional ones significantly improved the prediction of events at the beginning of a session and consequently boosted detection rates within short sessions, as we demonstrate in Section 5.8.1. Increasing the number of LSTM layers from one to two decreases false positive rates in longer sessions while maintaining similar detection rates, as we show in Section 5.8.2. In Section 5.10.1, we discuss why we are not further increasing the number of layers.

Output layer

The outputs from the bidirectional LSTM layers are then concatenated and fed to an additional linear hidden layer of size N^2_{hidden} with the commonly used rectified linear activation function. We added this layer to enable the network to learn more non-linear dependencies in a sequence. We found that by adding this layer, we are able to capture complex and rare behaviours and decrease false positive rates, as demonstrated in Section 5.8.3.

Finally, the output of this layer layer is fed to two output layers with M and S linear output cells. These produce two numeric vectors of size M and S

$$p_x^{i,j,k},\ k \in \{1,\dots,M\}, \qquad \sum_k^M p_x^{i,j,k} = 1$$
$$p_s^{i,j,l},\ l \in \{1,\dots,S\}, \qquad \sum_l^S p_s^{i,j,l} = 1$$

that describe the predicted probability distribution of $x^{i,j}$ and $s^{i,j}$ respectively.

The prediction loss for the state group is then given by the negative log-likelihood:

$$\text{lh}_x^{i,j} = \sum_{k=1}^M (1 - x_k^{i,j}) \cdot \log(1 - y_k^{i,j}) - x_k^{i,j} \cdot \log(y_k^{i,j})$$

with the size group loss being calculated in the same way. We calculate the total loss as the sum of the state loss and the size group loss. A visualisation of the prediction-making process is depicted in Fig. 5.4.

After the training, we use the network to determine the anomaly score of a given input session via the average of the predicted likelihoods, as this measure is independent of the session length:

$$\text{AS}^j = 1 - \sum_{i=1}^{N_j} \left(\exp(\text{lh}_x^{i,j}) + \exp(\text{lh}_s^{i,j}) \right) / N_j$$

An anomaly score close to 0 corresponds to a benign session with a very high likelihood while a score close to 1 corresponds to an anomalous session with

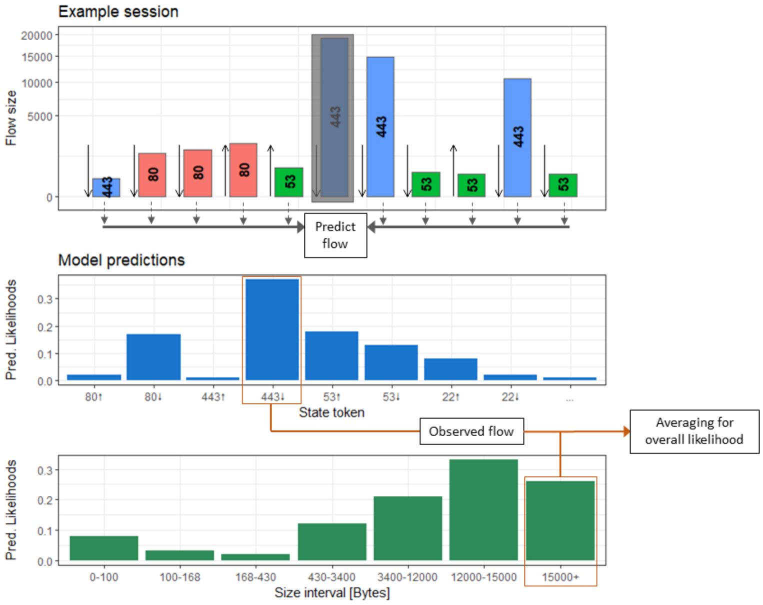


Figure 5.4: Visualisation of model prediction process.

events which the network would not predict in the context of previous events. We rescale all anomaly scores for better readability, however this does not influence their ordering.

5.4.5 Parameter selection and training

We now train CBAM and tune it to maximise its prediction performance. We train on a quad-core CPU with 3.2 GHz, 16 GB RAM, and a single NVIDIA Tesla V100 GPU, and we use minibatches of size 30 using the ADAM optimiser in PyTorch. Training a model can be achieved in under three hours.

We want to create a model that has sufficient parameters to capture complex flow dependencies, but is not overfitting the training data. For this, we split the available training data into a larger training and a smaller validation set. We then select two model configuration, one with a larger number of parameters and one with a smaller number. We then train the model for 500 epochs on the training set and observe whether the same loss decrease can be observed on the validation set. As long as the larger model is performing better than the smaller model and the validation loss is consistent with the training loss, we keep increasing the number of parameters, a standard practice to train deep learning models. The best performing parameters were $N^1_{\text{embed}} = 10$, $N^2_{\text{embed}} = N^3_{\text{embed}} = 5$ for the embedding layers, and $N^1_{\text{hidden}} = N^2_{\text{hidden}} = 50$ for the hidden layers.

To build a more powerful model without the risk of overfitting, we use a drop-out rate of 0.5 and a weight-decay regularization of 5×10^{-4} per epoch, as suggested by Hinton et al. [?]. To increase the training performance, we use an adaptive learning of 0.0003, which decays by a factor of 2 after each fifty subsequent epochs, as well as layer normalization. The values for the learning rate and weight decay were estimated in a similar procedure as the model size.

As we mentioned above, too large vocabularies can cause problems both for model training and event prediction. We achieved the best results for $M = 200$ for the available data and computational resources. The size of the size group was chosen smaller with $S = 7$, which improves detection capabilities without increasing anomaly scores for benign sessions too much. We found that a suitable value of $C = 4$ to describe different host types, which include servers, two types of client machines depending on the operating system, and auxiliary devices (printers, IP-phones, and similar). Host groups were determined for each dataset individually and the corresponding machines labelled manually. However, for larger datasets this process is easily automated by filtering for specific traffic events such as requests to Microsoft update servers.

	# cells	# parameters
Embedding layer	202/10/5	2055
LSTM-layer 1	50	6700
LSTM-layer 2	50	12700
Linear layer	50	2550
Softmax layer	202/10	10557
Total	**34562**	

5.4.6 Detection method

We use a simple threshold anomaly score to identify a session as malicious. We estimate the 99.9% quantile for benign sessions in the training data, which will then act as our threshold value T. By determining T from the training data, we control the expected false positive rate in the test data. Threshold values are determined for each dataset and each host within a dataset separately.

$$T_c : P[AS^j_{cj} \leq T_c] \leq 0.999$$

Finding an appropriate threshold value is a compromise between higher detection rates and lower false positive rates, and we chose this value to achieve false positive rates that are low enough for a realistic setting. We compare detection and false positive rates for different T in Section 5.6.1, and we give an outlook to more sophisticated detection methods in Section 5.10.

5.5 Datasets

5.5.1 Dataset assembly

The field of network intrusion detection has always suffered from a lack of suitable datasets for evaluation. Privacy concerns and the difficulty of posterior attack traffic identification are the reason that no dataset exists that contains realistic U2R/R2L (User-to-Root, Remote-to-local) traffic and benign traffic from a real-world environment [?]. To evaluate CBAM, we need both representative access attack traffic to test detection rates, and background traffic from a realistic environment to test false positive rates. To ensure that both criteria are met, we selected three modern publicly available datasets that complement each other: CICIDS-17 [?], LANL-15 [?, ?], and UGR-16 [?]. The CICIDS-17 dataset contains traffic from a variety of modern attacks, while the UGR-16 dataset's length is suitable for long-term evaluation. The LANL-15 dataset contains enterprise network traffic along with several real-world access attacks.

We train models with the same hyperparameters on each dataset to demonstrate the capability of CBAM to detect various attacks and perform well in a realistic environment.

CICIDS-17: This dataset [?], released by the Canadian Institute for Cybersecurity(CIC), contains 5 days of network traffic collected from 12 computers with attacks that were conducted in a laboratory setting. The computers all have different operating systems to enable a wider range of attack scenarios. The attack data of this dataset is one of the most diverse among NID datasets and contains SQL-injections, Heartbleed attacks, brute-forcing, various download infiltrations, and cross-site scripting (XSS) attacks, on which we evaluate our detection rates.

The traffic data consists of labelled benign and attack flow events with 85 summary features which can be computed by common routers. The availability of these features makes it suitable to evaluate machine-learning techniques that were only tested on the KDD-99 data.

The benign traffic is generated on hosts using previously gathered and implemented traffic profiles to make the traffic more heterogeneous during a comparably short time span, and consequently closer to reality. For our evaluation, we selected four hosts that are subject to U2R and R2L attacks, two web servers and two personal computers.

This dataset is generated in a laboratory environment, with a higher proportion of attack traffic than is normally encountered in a realistic setting. Consequently, we need to test on traffic from real-world environments to prove that CBAM retains its detection capabilities and low false alert rates.

LANL-15 dataset: In 2015, the Los Alamos National Laboratory (LANL) released a large dataset containing internal network flows (among other data) from their corporate computer network. The netflow data was gathered over a period of 27 days with about 600 million events per day [?, ?].

In addition to large amounts of real-world benign traffic, the dataset contains a set of attack events that were conducted by an authorised red team and are supposed to resemble remote access attacks, using mainly the *pass-the-hash* exploit. We selected this dataset to demonstrate that CBAM is able to detect attacks in a realistic environment with low false alert rates. We isolated traffic from ten hosts, with two being subject to attack events. Two of these hosts resemble server behaviour, while the other eight show typical behaviour of personal computers.

The provided red team events are not part of the network flow data and only contain information about the time of the attack and the attacked computer. Furthermore, not all of the attack events are conducted on the network level, so it is impossible to tell exactly which flows correspond to malicious activity and which do not. Therefore we labelled all flows in a narrow time interval around each of the attack timestamps as possibly malicious. As these intervals are narrow, identified anomalies likely correspond to the conducted attack.

UGR-16 dataset: The UGR-16 dataset [?] was released by the University of Grenada in 2016 and contains network flows from a Spanish ISP. It contains both clients' access to the Internet and traffic from servers hosting a number of services. The data thus contains a wide variety of real-world traffic patterns, unlike other available datasets. Additionally, a main focus in the creation of the

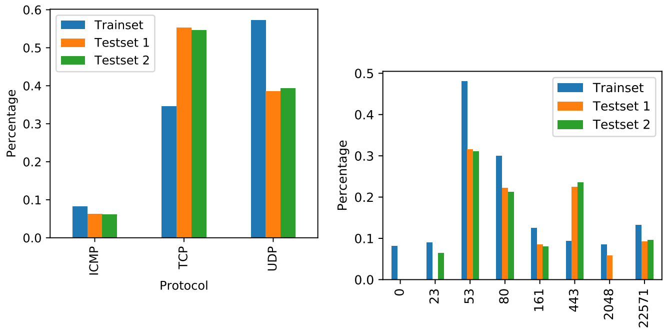


Figure 5.5: Temporal change in protocol and port usage over the different train and test intervals across selected servers in the UGR-16 dataset.

data was the consideration of long-term traffic evolution, which allows us to make statements about the robustness of CBAM to concept drift over the 163 day span of the dataset. For our evaluation, we isolated traffic from five web-servers that provide a variety of services.

5.5.2 Dataset split

We split our data into a test set and a training set. To resemble a realistic scenario, the sessions in the training data are from a previous time interval than sessions in the test data.

To evaluate detection rates on the CICIDS-17 data, we selected the four hosts in the data that are subject to remote access attacks, two web servers and two personal computers. We choose our test set to contain the known attack data while the training data should only contain the benign data. Due to the short timespan of the dataset, we have to train on traffic from all five days, with the test data intervals being placed around the attack. In total, the test set contains 14 hours of traffic for each host while the training set contains 31 hours of traffic. While the test set for the CICIDS-17 data covers a shorter timespan, it contains more traffic due to voluminous brute-force attacks.

For the LANL data, the test set stretches approximately over the first 13 days with the training data spanning over the last 14 days. The unusual choice of placing the test set earlier the training set was made because the attacks occur early in the dataset. However as the training and test are contained in two non-overlapping intervals, a robustness evaluation is still possible.

Dataset	hosts	sessions in training set	sessions in test set	length
CICIDS-17	4	24128	32414	5 days
LANL-15	10	89480	76984	27 days
UGR-16	5	65000	480018	163 days

Table 5.2: Summary of the amount of traffic extracted from each dataset.

To test long-term stability and robustness of CBAM against concept drift, we split the UGR-16 data into one training set interval and two test set intervals, for which we can compare model performance. The training set interval stretches over the first month, with the first test set interval containing the sessions from the following two months, and the second test set interval containing the last two months. We then isolated traffic from five web-servers that provide a variety of services that show behavioural evolution. Fig. 5.5 depicts the changes of these servers in terms of protocol and port usage over the different intervals.

We chose our training data to contain about 10 000 sessions per host if possible. A summary of the amount of data in the training and test data for each dataset can be found in Table 5.2.

5.5.3 Sample imbalance and evaluation methodology

Most NID datasets include attack events from both low volume access attack classes as well as attacks like DoS or port scans which generate a large number of events. If reported detection rates do not distinguish between different attacks or attack classes, performance metrics will be dominated and potentially inflated by DoS and probing attacks. Similarly, detection results are often given in terms of precision and recall or F-measures, which are sensitive to the specific dataset balance of the majority and minority classes in a dataset. Since the ratio of attack traffic is inflated in general NID-datasets, these measures are not suitable for model comparison.

We evaluate CBAM using simple detection true positive and false positive rates, which are independent of the dataset balance. We also distinguish detection rates for different attacks, and assess overall performance by averaging these rates over attack classes rather than overall number of attack events. Since there is no agreed upon value for a suitable false positive rate in network intrusion detection, we compute ROC-curves to display the detection rates in dependence of the false-positive rate and report overall AUC-scores (*Area under Curve*), which describe the separation of benign and anomalous traffic. We use these for comparison with other models, as this measure is fairer than point comparisons. The evaluation procedure is supported by several NID evaluation surveys [**?**, **?**].

Some researchers propose cost-based evaluation metrics by assigning false alerts and missed intrusion attempts a cost-value and tuning the detection threshold to minimise the expected cost, such as done by Ulvila and Gaffney [**?**]. Such a metric is however strongly dependent on the observed ratio of attack to benign traffic, which is strongly inflated in NID-datasets, and requires operational information to assign costs to a false alert or intrusion. This evaluation works well in specific cases such as DoS-attacks or cryptojacking, where server-downtime costs and attack volume are generally quantifiable, but is not applicable in cases where this information is not available or well defined [**?**].

Training classifiers on imbalanced dataset can affect their performance, both due to the imbalanced ratio of attack to benign traffic and the imbalance between several attack classes. Some methods have been proposed to augment or synthetically inflate minority samples for attack traffic [**?**]. As an anomaly-detection method, CBAM is however trained in a self-supervised way strictly on benign traffic, with no attack traffic being present in the training data. The training stage is therefore independent of the minority class ratio in a given dataset and does not require specific balancing methods. In the evaluation stage, the above described steps apply for both classification- and anomaly-detection-based methods.

5.6 Detection performance

We now demonstrate that we can build an accurate and close-fitting model of normal behaviour with CBAM. We train models for each dataset separately, but without any change in the selected hyperparameters, i.e. number of hidden cells, vocabulary size, learning rate etc.

As described above, we estimate detection rates using traffic of various remote access attacks in the CICIDS-17 dataset. Table 5.3 describes the number of sessions present for each attack class.

	FTP-BF	SSH-BF	Web-BF	SQL-Inj.	XSS	Heartbleed	Infiltr.
# Sessions	243	210	88	8	41	4	17

Table 5.3: Number of sessions for each attack class in the CICIDS-17 dataset

5.6.1 CICIDS-17 results

Table 5.4 and Fig. 5.6 depict anomaly score distributions and detection rates for traffic from seven different types of attacks.

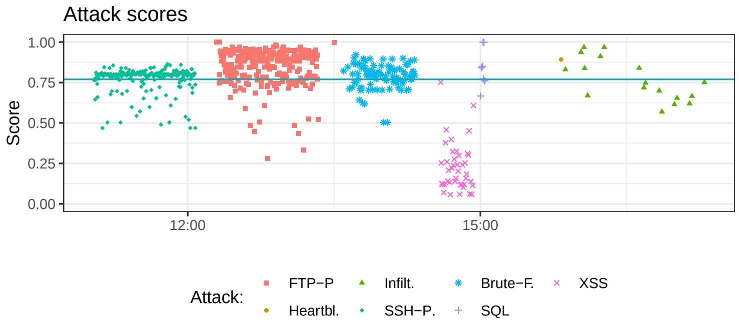


Figure 5.6: Score distribution for access attacks contained in the CICIDS-17 dataset.

Most notable is that scores from all attacks except cross-site scripting (XSS) are significantly higher distributed than benign traffic, with median scores lying between 0.75 and 0.89. Detection rates with our chosen threshold of $T = 0.77$ are highest for Heartbleed attacks (100%), followed by FTP and SSH brute-force attacks and SQL-injections, where 91%, 74%, and 75% of all affected sessions are detected. Detection rates are lowest for XSS and Infiltration attacks. The overall detection rates we achieve are in a similar range as most unsupervised methods in Nisioti et al.'s evaluation [?], but with significantly better false positive rates.

XSS and infiltration attacks cause the victim to execute malicious code locally. Heartbleed and SQL injections on the other hand exploit vulnerabilities in the communication protocol to exfiltrate information, and are thus more likely

	Anomaly scores (T=0.77)			Detection	Shallow
	min	max	median	rates [%]	LSTM
Brute-force Web	0.50	0.92	0.80	0.66	0.28
FTP-Patator	0.28	1.00	0.82	0.91	0.38
Heartbleed	0.89	0.89	0.89	1.00	0.0
Infiltration	0.57	0.97	0.75	0.41	0.0
SQL-injection	0.67	1.00	0.84	0.75 2	0.21
SSH-Patator	0.47	0.86	0.80	0.74	0.67
XSS	0.06	0.75	0.20	0.00	0.0

Table 5.4: Anomaly score distributions and detection rates at threshold T for known-malicious sessions in the CICIDS-17 dataset, as well as detection rates for a less complex benchmark model (Fig. 5.17).

to exhibit unusual traffic patterns, visible as excessively long SQL-connections or completely isolated TCP-80 flows for SQL-attacks, or unusual sequences of connections initiated by the attacked server during Heartbleed attacks.

Brute-force attacks on the other hand cause longer sequences of incoming connections to the same port of a server, in this case to port 21 for FTP, 22 for SSH, and 80 for web brute-force. Especially for port 80, such sequences are not necessarily unusual, which explains the difference in detection rates between web brute-force, which CBAM does not detect reliably, and FTP and SSH brute-force, which are detected at a higher rate. Depending on how much benign traffic the particular sessions are overlayed, the estimated anomaly scores can vary. Brute-Force attacks are not low in volume, and spread over many sessions since we introduced a maximum session length. For these types of attack, CBAM therefore only has to flag a smaller percentage of malicious sessions the attack generates to detect anomalous behaviour.

Fig. 5.7 provides *ROC* (Receiver operating characteristic) curves for each attack type. As seen, for Heartbleed, FTP brute-force, SQL injection, and infiltration attacks, CBAM starts detecting attacks with close to zero false positives.

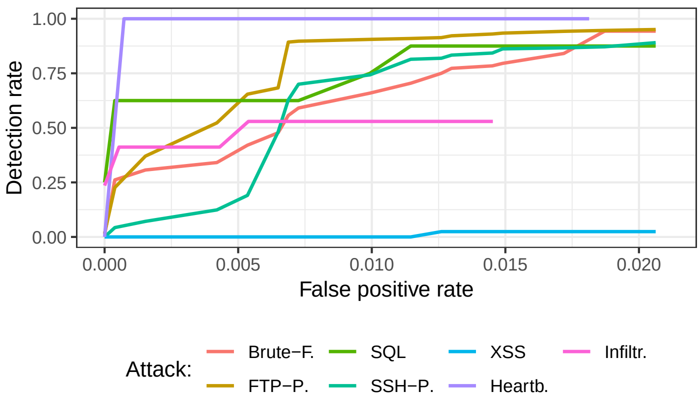


Figure 5.7: ROC-curves for different attack types in the CICIDS-17 dataset.

5.6.2 LANL results

We now examine whether CBAM is able to detect actual attacks in real-life traffic from the LANL-15 dataset.

As described in Section 5.5.1, we do not have labels for malicious flows in the LANL-15 data. Instead, attacks are described by narrow intervals surrounding conducted malicious activity. These intervals inevitably contain benign activity too. However, as the intervals are narrow and we saw that benign sessions only rarely receive high anomaly scores, a session with a high anomaly score is likely to be associated with a malicious event. Of the hosts in the dataset we selected for evaluation, hosts C2519 and C754 are subject to the red team attacks. The red team activity is spread over three attack intervals **A1**, **A2**, and **A3**.

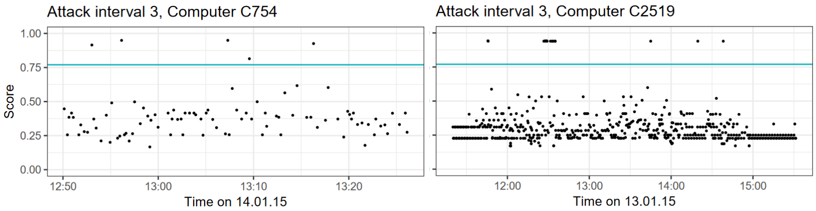


Figure 5.8: Computed scores for the third attack interval in the LANL data, along with our detection threshold.

Sessions in **A1** and **A2** have similar scores as other benign traffic, with no sessions receiving remarkably high scores. It is both possible that CBAM did not identify the malicious traffic, or that the activity in these intervals was purely host-based and did not generate any traffic.

Interval **A3** is more interesting, containing 15 sessions for host C2519 and 5 sessions for host C754 that have high anomaly scores, as depicted in Fig. 5.8.

For host C2519, each session with a high anomaly score consists of a single TCP-flow on port 445, which is usually reserved for a Microsoft SMB service. The anomaly of these sessions becomes apparent when we compare them to other sessions that contain TCP-flows on port 445.

	Proto	SrcAddr	Sport	DstAddr	Dport	Prob_End
1	tcp	C20473	N345	C2519	445	0.03
2	tcp	C2519	445	C20473	N345	0.55

Table 5.5: Exemplary session with SMB-traffic, host C2519, along with the estimated probability of the session to end.

All other sessions contain at least two subsequent flows. The model, in expectation of other following flows, assigns a very low probability to the sessions ending after a single flow. Since the analysis of the identified sessions supports their anomaly score, we believe it is very likely that these events correspond to the conducted malicious activity.

5.6.3 How attacks affect flow structures

We now examine in more detail why modelling sequences of flows is effective to detect access attacks, and how these attacks alter common flow structures. Unfortunately, the CICIDS-17 dataset, and to our knowledge all other NID-datasets, do not contain sufficient ground truth information about included attack traffic, so this analysis is based on empirical domain-knowledge of similar attacks as well as the traffic itself.

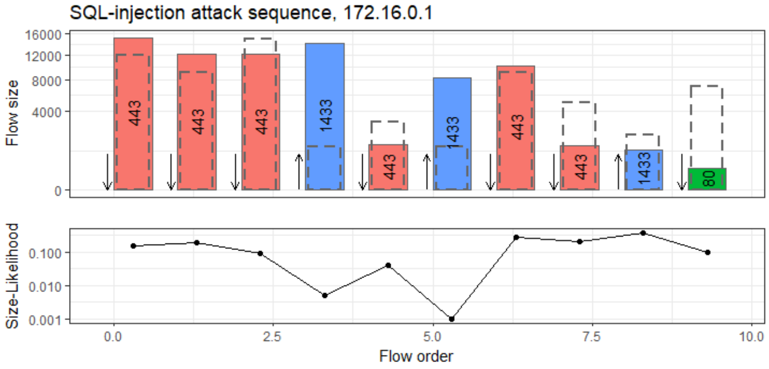


Figure 5.9: Flow-sequence in an SQL-injection attack with predicted size likelihood (log-scale). Arrows indicate flow directions (down=incoming, up=outgoing).

Fig 5.9 shows a session in the CICIDS-17 data that corresponds to a SQL-injection attack on host 172.16.0.1, a Ubuntu web server. Depicted is the order of the flows along with their direction, the destination port and the size of the flow. Dashed rectangles indicate the most likely flow size as predicted by CBAM. Below are the likelihoods of the actually observed flow sizes on a log-scale, which determine the anomaly-score of the session.

SQL-requests from a web-server typically consist of verification of user credentials or the retrieval of specific content on a webpage. In an SQL-injection, SQL-code is injected into a HTTP-request that forces the server to retrieve, modify or forward additional content from an SQL-database, which can significantly increase the size of the corresponding SQL-request.

The sequence of flows in Fig. 5.9 overall resembles regular incoming HTTP requests accompanied by corresponding outgoing SQL-requests from the server. However, Fig. 5.9 clearly shows that the sizes of two of the SQL-connections on port 1433 are magnitudes larger than CBAM is predicting based on the context of the surrounding flows, which is likely caused by the injection attack. This results in a very low likelihood of the observed flow sizes and a high anomaly score for the whole session.

Fig 5.10 depicts a session that corresponds to an infiltration attack on host 192.168.10.8 in the CICIDS-17 data. Again, the figure depicts the order, direction, size and destination port of the flows, along with predictions of the most-likely sizes (dashed rectangles) and the overall likelihood of the actually observed flows.

This sequence does not resemble regular behaviour typically encountered on this host. DNS flows on port 53 are typically followed by HTTP flows on port 80 or 443, to which the model assigns a very high likelihood after the first 4 flows. However, this session contains excessively many consecutive DNS flows, which are interrupted by only one HTTP flow. Correspondingly, the likelihood for the excessive DNS flows as well as the overall session likelihood is low.

It is not completely clear how the infiltration attack triggers this abnormal behaviour. Possibly, the infiltration software is trying to retrieve the current address of a C&C-server via DNS.

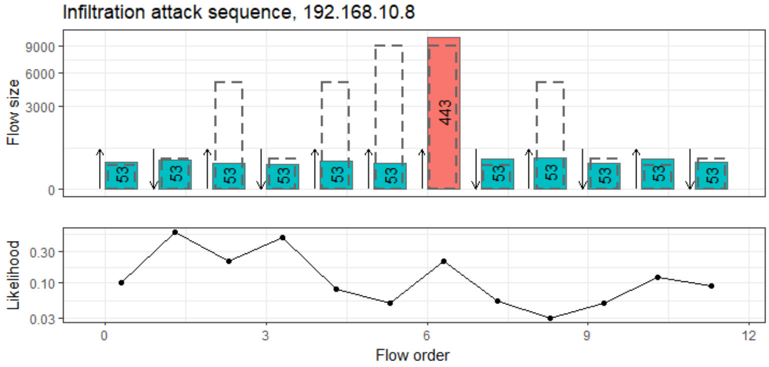


Figure 5.10: Flow-sequence in an infiltration attack with port-direction likelihood (log-scale).

5.6.4 Runtime performance

CBAM contains around 35 000 parameters, which is relatively lightweight for deep learning models. The processing of a session of ten flows takes around $23ms$ on our setup, which is far shorter than the average length of $5.6s$ of a session. In a similar comparison, our setup can process one day of activity ($\approx 15\,000$ sessions) of a web server in the UGR-16 dataset in $95s$.

Considering these runtime numbers, the necessary rate of recorded flows to overwhelm our setup would need to exceed 434 flows/second. The largest rate observed for Brute-Force attacks in the CICIDS-17 dataset is 23 flows/second.

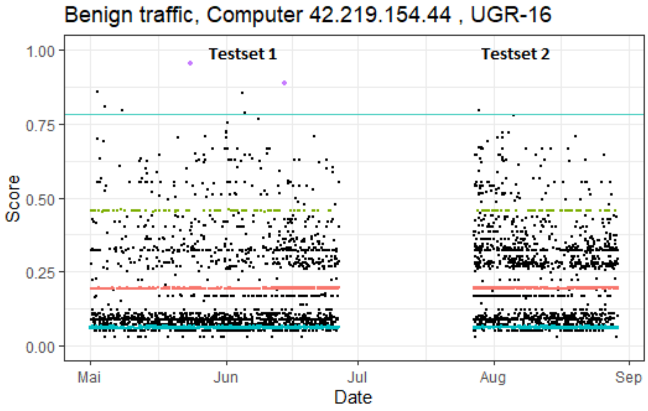
5.7 Benign traffic and longterm stability

5.7.1 UGR-16 data

We conduct the main validation of the longterm stability of CBAM on benign traffic in the UGR-16 dataset, which contains real-world traffic and spans several months. For this, we split the test data into two disjoint sets that span from May-July and August-September while being separated by one month. We then look at the quantiles and visual distribution of session scores in each test set and assess whether the score distributions and number of false positives changed as evidence on concept drift in the traffic. Fig. 5.11 depicts the score distribution of benign sessions for each dataset in the corresponding test sets.

As visible in the plot, the centre of the score distribution is concentrated very well in the lower region of the $[0, 1]$ interval, with about 50% of all sessions receiving scores in the region between 0.1 and 0.25. High scores are rare, with only very small percentages exceeding our chosen detection threshold of $T = 0.76$.

This is also reflected by the corresponding table that describes score distributions for all 5 hosts in the UGR-16 data. On average less than 0.15% of all



	test set 1			test set 2		
	50%	90%	Pr(>T)	50%	90%	Pr(>T))
42.219.153.32	0.21	0.39	0.01%	0.22	0.38	0.01%
42.219.155.189	0.12	0.20	0.01%	0.10	0.21	0.03%
42.219.155.128	0.24	0.44	0.63%	0.19	0.43	0.41%
42.219.155.4	0.13	0.34	0.10%	0.17	0.39	0.23%
42.219.154.44	0.11	0.32	0.13%	0.11	0.28	0.11%

Figure 5.11: Anomaly score distributions for benign traffic in the UGR-16 data, along with an exemplary distribution plot for a selected host.

assumed-benign sessions exceed the threshold, which would translate to fewer than ten false-alerts over the span of four months on a host with similar activity rates.

Differences in the score distributions for the two test sets are quasi non-existent. The core of the distributions are very stable, with the score quantiles differences being less than 0.03. There are some differences in the observed false positives, but the available sample size is not large enough to make any statements on any systematic differences.

A clear banding structure is visible in the plotted distributions, with most session scores being very concentrated on narrow intervals. These scores represent frequently reoccurring activities that generate very similar traffic sequences. Fig. 5.11 shows that these banding structures remain virtually unchanged over several months and carry over from test set 1 to test set 2.

We coloured three of these bands in Fig. 5.11 at different levels as well as two of the observed false-positives, which we are now examining closer. Fig. 5.12 depicts the corresponding dominant session pattern that is present in each band along with the predicted likelihood for each flow. Again, the figure depicts the order, direction, size and destination port of the flows, along the overall likelihood of observed flows. For clarity, we omitted the predictions of the most-likely flow sizes (dashed rectangles).

The two lower bands, blue and red located at $AS = 0.061$ and $AS = 0.18$, represent simple and frequent HTTP- as well as corresponding NoSQL-requests and SSH activity by the server. These sessions are therefore predicted with high accuracy.

The green band at $AS = 0.45$ represents more complex and longer sessions

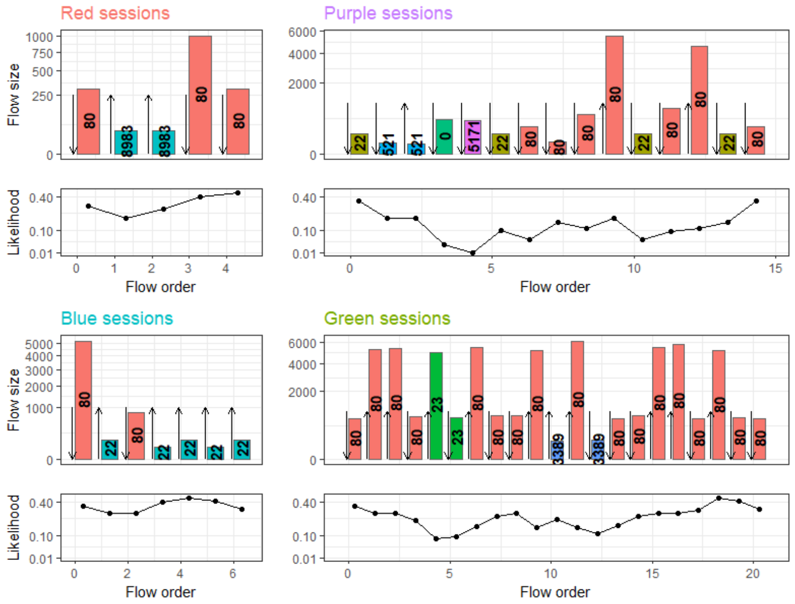


Figure 5.12: Sessions corresponding to score banding structures in Fig. 5.11, with predicted likelihoods (log-scale).

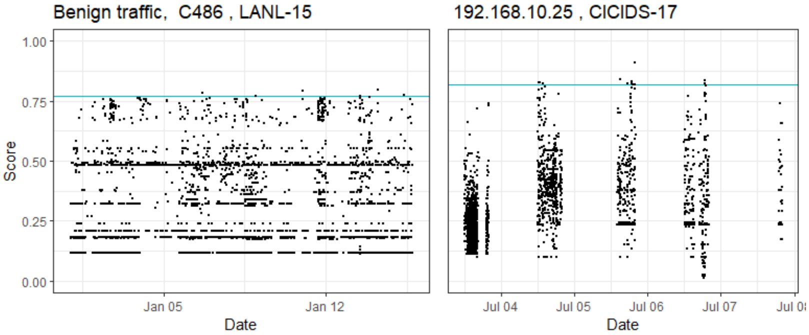
that involve both incoming and outgoing HTTP-connections as well as TelNet and RDP connections. The size and order of the flows in these sessions is less deterministic than the activity in the red and blue bands. This activity is also less frequent, which explains the less accurate predictions by CBAM. The model however still recognises these sessions and is able to predict flow state and size with a non-vanishing probability, which keeps the overall session score bounded.

The two purple-coloured sessions likely represent server inspection activity, involving activity on port 0, SSH-sessions and activity on uncommon ports. This type of activity is very rare on this server and appears less deterministic than other more common activity. CBAM therefore fails to recognise the session structure and is not able to assign non-vanishing probabilities to several flows, which decreases the overall session likelihood and results in a high anomaly-score. We are not aware how often server are subject to inspections and whether this would present a problem in operational deployment. However, it seems feasible that resulting false-alerts could be linked to this administrative activity automatically or by a security analyst.

5.7.2 CICIDS-17 and LANL-15 results

We now look at the structure and stability of anomaly scores for benign traffic in the LANL-15 and CICIDS-17 datasets. The plots and tables in Fig. 5.13 depict the score distribution of presumably benign sessions in both datasets as well as describe the 50% and 90% quantiles and false-positive rates for each host. Again, score distributions for both datasets are concentrated well in the lower region of the $[0, 1]$ interval. For both datasets, the median lies between 0.06 and and 0.29.

For the LANL-15 data, we observe the same banding structure as in the UGR-15 data, with most sessions being concentrated in these bands. This banding is however far less pronounced in the CICIDS-17 data, with the majority of session



	50%q	90%q	Pr(>T)
C10047	0.06	0.34	0.053%
C443	0.06	0.33	0.119%
C2519	0.19	0.25	0.064%
C13845	0.38	0.63	0.40%
C486	0.12	0.48	0.16%
C7379	0.09	0.38	0.11%
C754	0.22	0.40	0.08%
C7564	0.06	0.38	0.11%
C9020	0.10	0.43	0.19%
C9676	0.29	0.51	0.19%

	50%q	90%q	Pr(>T)
10.25	0.25	0.49	0.09%
10.50	0.11	0.50	0.17%
10.51	0.38	0.63	0.42%
10.8	0.25	0.56	0.064%

Figure 5.13: Anomaly score distributions for benign traffic in the LANL-15 and CICIDS-17 datasets.

scores here being dispersed to a greater extent. This suggests that the traffic generation process for this dataset relies far less on reoccurring rigid activities than we observe in real-life data, which however does not seem to deteriorate the prediction performance of CBAM.

5.7.3 Importance of training data size

Host C13845 in the LANL-15 and host 192.168.10.51 in the CICIDS-17 data are an exception from the above observations, with their median anomaly score being 0.38 each and their estimated false-positive rates being 0.4% and 0.42%, which significantly exceed the average of 0.1%.

When examining host 192.168.10.51, we notice that it produced less traffic than other hosts in the CICIDS-17 data. Due to this fact, the training dataset only contains 3 096 sessions or 36 989 flows for this host, compared to about 10 000 sessions or 115 000 for host 192.168.10.25.

For hosts C13845, we see a similar picture. Because the host is less active than others in the dataset, the training data contains only 728 sessions or 2 423 flows for this host, compared to 6 013 sessions for the host with the next fewest training sessions.

This suggests that traffic on these hosts are not necessarily harder to predict for CBAM, but that the lack of sufficient training data prevents CBAM from learning traffic patterns for these two hosts effectively. To verify this, we examined how many sessions are necessary in the training phase to achieve similar false

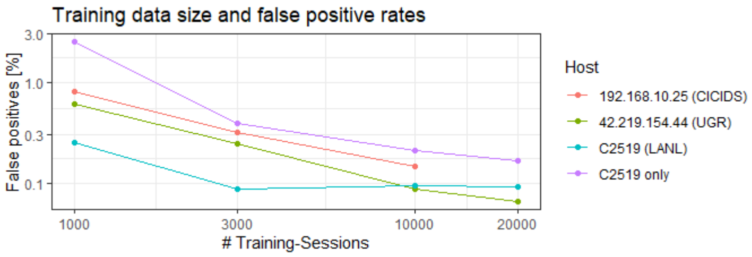


Figure 5.14: Influence of number of sessions in training data for benign traffic modelling accuracy.

positives at a given anomaly threshold. We selected the hosts with the most sessions for each dataset, and reduced the number of training sessions from 10 000 to 3 000 and 1 000. We then trained models with otherwise similar settings and compared how many additional sessions exceed the anomaly threshold. For UGR-16 and LANL-15, we examined if increasing the number of training sessions to 20 000, which was not possible for the CICIDS-17 data.

Fig. 5.14 depicts the corresponding false-positive rates for each host. False-positive rates for the UGR-16 and the CICIDS-17 hosts are already significantly affected when only trained on 3 000 sessions, and increase further when only 1 000 host sessions are available during training. Increasing the number of sessions to 20 000 however does not seem to have an effect to further improve the model.

For the host in the LANL-data, this effect is far less pronounced, and false-positives at 3 000 sessions are similar to the ones at 10 000 and 20 000. CBAM apparently is able to learn flow predictions sufficiently from similar hosts in the dataset without depending on sessions specifically from the selected host. When we train CBAM exclusively on sessions from host C2519 without data from other hosts, the same deterioration of model prediction can be observed.

5.8 Benefit of increased model complexity

A significant part of the conducted work was concerned with improving the given network design to address insufficient predictions for several traffic phenomena and boost overall model performance. We now outline several key-steps in the design process and how they improve performance.

5.8.1 Bidirectionality for better session context

The usage of bidirectional LSTM layers compared to unidirectional ones significantly improved the prediction of events at the beginning of a session and consequently boosted detection rates within short sessions. Fig. 5.15 demonstrates this in a detailed manner: Displayed is a short session of 4 flows containing FTP and HTTP activity on host “42.219.153.32”. On the right side are the predicted likelihoods of FTP and HTTP states for each flow in the session, with the the blue bars corresponding to predictions by the forward layer, while the red bars display the backwards direction and the green bars display the likelihoods after aggregating both predictions.

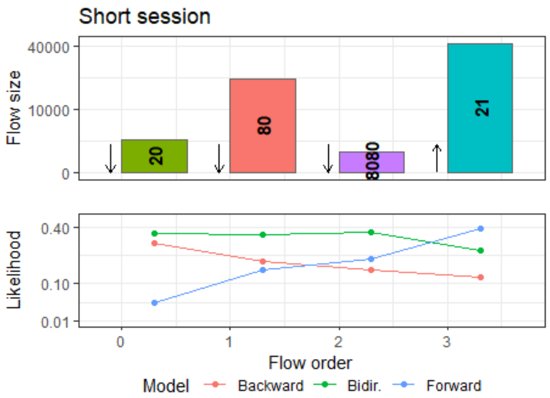


Figure 5.15: Common short session and the flow likelihoods predicted by each dirctional model.

Fig. 5.15 demonstrates this in a detailed manner: Displayed is a short session of 4 flows containing FTP and HTTP activity on host “42.219.153.32”. On the right side are the predicted likelihoods of FTP and HTTP states for each flow in the session, with the the blue bars corresponding to predictions by the forward layer, while the red bars display the backwards direction and the green bars display the likelihoods after aggregating both predictions.

When only relying on the forward direction, for the first two flows the predicted likelihoods are less than 0.07 each. The last two flows of the session are however predicted well with high likelihoods over 0.3. Because the session is short, the inaccurate predictions for the first two flows decrease overall likelihood of the session to 0.18 and the corresponding anomaly score to $AS = 0.73$, which is just below the anomaly threshold, even though this type flow sequence is quite common in the UGR-16-dataset. In a similar manner, this applies for the backward direction with the likelihoods of the last two observed flows being 0.02 and 0.03 respectively.

		Likelihood of flows 1&2		FP rate [%]	
		unidir	bidir	unidir	bidir
UGR-16	All sess.	0.13	0.27	0.31	0.12
	sess.¡5 flows	0.19	0.41	1.6	0.09
CICIDS-17	All sess.	0.09	0.29	0.37	0.18
	sess.¡5 flows	0.05	0.30	1.7	0.13

Table 5.6: Average likelihood of first two flows in a session and false-positives for uni- and bidirectional model.

The cause for these phenomena is that the start of a session can differ significantly for different activities, and the LSTM-layer needs some context before recognising the specific activity and make corresponding predictions. In short sessions, the lack of accurate predictions in the first flows can then dominate the anomaly-score of the whole session.

By adding a bi-directional layer, we are able to provide context for these initial flows in a session as well by looking at later flows first. The green bars in Fig.

5.15 displays this: By basing predictions both on the output of the forward- and the backward-layer, the bidirectional model is able to predict flow likelihoods significantly better and thus assign the session a much lower anomaly-score.

Table 5.6 displays how much we could decrease false-positive rates by replacing the unidirectional LSTM layer with a bidirectional one. Overall, the false-positives decreased by 61% for the CICIDS-17 dataset, and by 52% for the UGR-16 dataset. More strikingly, when only looking at short sessions that contain less than 5 flows, we were able to reduce false-positives by 94% and 87% respectively.

5.8.2 Additional layers for complex session modelling

The inclusion of a second LSTM-layer as well a subsequent linear layer allows CBAM to capture more complex behaviour in long sessions as well as remember rare behaviour more quickly. It also increased the average predicted likelihood for flows overall.

To examine the benefit of the described model depth, we compare it to a more shallow version that lacks the second LSTM- and linear layer, as depicted in Fig. 5.17, which was trained under otherwise similar conditions. Overall detection rates for this model can be found in Table 5.4, while score distributions can be found further down in Table 5.7. Here, we examine in more detail how the increased model depth allows better predictions for complex sessions.

Fig. 5.16 displays two different types of activities, A and B, which are common in the CICIDS-17 data. The structure in these sessions can be observed frequently with only minor variations. Consequently, the sessions are predicted well by both the original and the more shallow model.

However, traffic from two or more activities can sometimes occur simultaneously and thus get grouped into the same session. Fig. 5.16 shows how the traffic from activity A and B are overlapping in a session, which makes the structure in the session more complex to predict.

The displayed likelihoods show predictions by the shallow model are accurate for flows at the beginning of the session well, but deteriorate once encountering flows from activity B. Prediction accuracy by the more complex model is also decreasing, but remaining on a sufficient level to assign this session an anomaly score of $AS = 0.51$, compared to $AS_S = 0.79$ for the shallow model. When looking at the activation in the LSTM-memory cell, we see that similar neurons as in activity A are activated at the beginning of the session, which shifts during the course of the session and resembles a more similar activation as in activity B at the end.

The improvements achieved by adding these additional layers could suggest that increasing the number of layers even further will decrease false positive rates even further, which we discuss in Section 5.10.

5.8.3 Comparison with simpler models

In this section we aim at studying whether the higher complexity of an LSTM network is necessary for the task of detecting contextual network anomalies, or whether simpler baseline methods can achieve the same results. For compar-

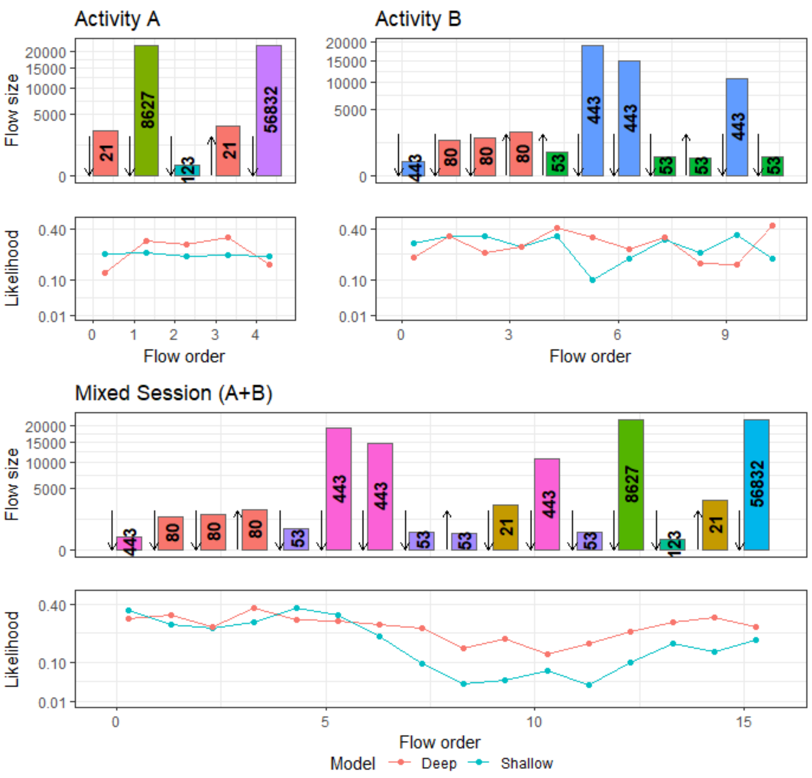


Figure 5.16: Predictions for two activities in isolated sessions and in a mixed session.

ison purposes, we implemented a first-order Markov Chain (MC) and a Non-Deterministic Finite Automata (NDFA) model. Both methods are widely used in sequence modelling, and have been applied successfully to security problems [?, ?]. In contrast to LSTMs, Markov Chains have no memory past the last event while NDFAs can distinguish between different types of sequences via state-merging, and give corresponding transition probabilities.

Similar to our LSTM model, Markov Chains and Finite Automata predict state transition probabilities, which is why we can employ the same anomaly score computation. However, computational costs increase quadratically with the number of states, and a separation of state vocabularies is not possible. We restrict both comparison models to the above described port-direction states. Models and detection rates were determined on the CICIDS-17 dataset.

Table 5.7 shows distribution characteristics of benign and malicious sessions for our shallow LSTM-model, the Markov Chain model, and the NDFA. It shows that CBAM outperforms these baseline methods, but also that the automata performs better than Markov Chains. While the Markov Chain is practically not able to make any distinction between malicious and benign traffic, the automata model shows some, albeit limited ability to identify anomalous sessions, mainly for the three types of brute-force attacks. This order shows the importance of sequence memory for contextual anomaly detection, and confirms our previous comparison of the suitability of Markov Chains and NDFAs for network intrusion detection [?].

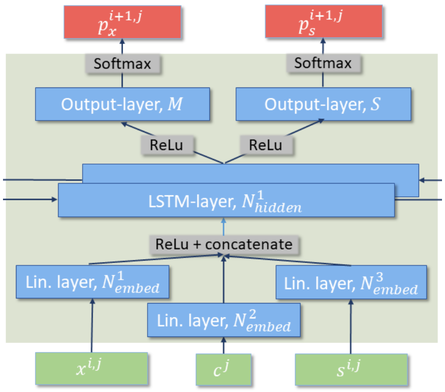


Figure 5.17: Architecture of the shallow LSTM-model.

	shallow LSTM	Markov MC	NDFA
Ben. 50%q	0.22	0.61	0.55
Ben. 90%q	0.55	0.81	0.86
Ben. 99%q	0.73	0.89	0.96
Mal. 50%q	0.70	0.60	0.68
Mal. 90%q	0.85	0.83	0.89
Mean AUC	0.86	0.53	0.64

Table 5.7: Score distributions for simpler models.

5.9 Related work

The application of recurrent neural networks to network intrusion detection has risen in popularity recently. LSTM-models for web attack detection, such as by Yu et al. [?], improve detection rates of simpler preceding models such as Song et al. [?]. They rely on deep packet inspection, and are often targeted at protecting selected web-servers rather than network-wide, due to a lack of computational scalability and increasing traffic encryption. Methodologically, vocabularies are created from string sequences with well-known NLP methods, while CBAM provides a new vocabulary-construction method suitable for traffic metadata.

The majority of LSTM-based metadata approaches rely on labelled attack data for classification, and do not have the scope of anomaly-based models to detect previously unseen attacks. A prominent example of this comes from Kim et al. [?], who classify flow sequences based on 41 numeric input features. Anomaly-based approaches, such as ours, mostly rely on iterative one-step ahead forecasts, with the forecasting error acting as the anomaly indicator. This is for instance done in GAMPAL by Wakui et al. [?], who use flow data aggregation as numerical input features, which are computationally easier to process, but cannot encapsulate high-level information such as the used protocol, port, or direction. These models are best used for detecting high-volume attacks. Apart from our work, only Radford et al. [?] create event vocabularies from flow protocols and sizes. We use a more sophisticated model in terms of stacked recurrent layers and embeddings for more input features, which results in higher detection rates, as demonstrated in see Section 5.8. The HCRNNIDS model by Kahn provides an interesting adaption of hybrid convolutional recurrent networks typically used in video modelling to intrusion detection [?] with promising results. In comparison to CBAM, this model is applied to individual flow features rather than flow sequences, and is trained as a classifier rather than an anomaly-detection model.

Encoding methods are increasingly used in combination with LSTM networks to create embeddings of packet or flow sequences, such as done by Zhong et al. [?] for anomaly detection. Zhou et al. [?] use embeddings to facilitate anomaly-detection that is robust against dataset imbalances. Liu et al. [?] use embeddings to augment and inflate minority class data samples for the same purpose.

Berman et al. [?] have surveyed recent deep-learning techniques for network intrusion detection as well as other cyber-security applications. They assess that many recurrent methods are state-of-the-art, but do not reach a conclusion whether they perform better than convolutional or generative methods.

Notable work outside of network traffic includes Tiresias [?], an LSTM model for security event forecasting with great accuracy, and DeepLog [?], an LSTM network to learn a system's log patterns (e.g., log key patterns and parameter values) from normal execution. The design of Tiresias has similarities to ours, but the scope of the model is attack forecasting rather than intrusion detection, and relies on both different input data in the form of IDS logs as well as different evaluation metrics. DeepLog is combined with a novel log parser to create a sequence of symbolic log keys, which is then also modelled using one-step forecasting. The authors achieve good detection results in regulated environments such as Hadoop with limited variety of events (e.g., 29 events in Hadoop). Here, CBAM goes further by being applied to a much more heterogeneous data source and creating a more than 30 times larger vocabulary. Han et al. [?] have recently proposed a deep graph-net based anomaly-detection method for provenance based data that demonstrates how effective neural anomaly-detection methods at detecting unkown intrusions.

5.10 Limitations and evasion

5.10.1 Limitations

CBAM is an initial application of short-term contextual modelling on network traffic that demonstrates the potential of contextual traffic models for intrusion detection. Although we use a relatively simple model with few, but carefully selected input features, we outperform sophisticated methods while retaining low false positive rates. The detection rates are to be taken with care as the available access attack data is small, synthetic and contains just a limited number of attack classes. The detection rates in the cross-evaluation on a real-world access attack in the LANL-15 data gives us confidence that CBAM's performance is reproducible in real-world scenarios, but additional data is required for an ultimate conclusion.

A frequently asked question concerns whether low false-positive rates carry over from the synthetic background traffic in datasets such as CICIDS-17 to real-world scenarios [?]. We believe that this was sufficiently demonstrated by the long-term evaluation and the observed score stability on the UGR-16 real-world dataset.

The improvements achieved by adding additional layers could suggest that increasing the number of layers even further will decrease false positive rates even further, and is certainly worth exploring in future work. However, as discussed in Section 5.7.1, the current main source for false positives are rare activity events which are not contained in the training data and are therefore not be recognised by the model. To make significant reduction in the false positive rate, we would need to train on datasets spanning more computers or over longer time periods. We are however aware of the difficulties involved in creating datasets for NIDS evaluation.

5.10.2 Evasion and resilience

Evasion tactics and corresponding model resilience against them have been a concern in the development of NIDS. We specifically focused on short-term sequential anomalies as they are often an unavoidable by-product of attack sequences, and it is thus very difficult for an attacker to perturb attack sequences that rely on a specific exploit without pre-existing control over the victim device or other network devices. We therefore believe that CBAM is relatively robust against evasion. However we identified potential improvements for future work.

A specific evasion tactic that has been discussed extensively in the context of machine learning is model poisoning in the training/retraining phase. A great difficulty for the attacker is the fact that CBAM uses sequences of symbolic events rather than continuous parameters. The introduction of a gradual shift is therefore not possible in a direct way as the alteration of individual events would look anomalous straight away. Furthermore, it is normally not possible for an attacker to alter individual events significantly without pre-existing control over network devices or specific exploits, i.e. the change of the port or size would normally cause an error in the communication. It is thinkable that an attacker could increase the predicted probability of specific events patterns more gradually by overlaying traffic stemming from 3rd party devices. However, the attacker would either need control of these devices or the ability to monitor traffic to the victim device in real-time, both of which is usually not available. We also showed in Section 5.7 that short-term contextual traffic patterns remain stable over several months, which means that retraining of CBAM is only necessary at a low rate and attackers will have to wait for a long time to execute successful model poisoning.

An issue we encountered is the overlay of malicious and benign traffic. Currently, the existence of known traffic patterns in a session can deplete the overall anomaly score of a session. A potential evasion tactic could therefore try to conceal an attack behind benign communication on the victim device, an already common approach for C&C communication. Possible improvements for this issue are a refined notion of a session that groups related traffic better, and a better scoring method that identifies smaller anomalous sequences in an otherwise normal sequence of flows. Additionally, developing more sophisticated detection methods from the computed scores may boost detection rates.

5.11 Conclusion

CBAM presents a new and promising angle to anomaly-based intrusion detection that significantly improves detection rates on the types of network attacks with the lowest detection rates. We use an anomaly-based approach that does not rely on specific notions of attack behaviours, and is therefore better suited at detecting unknown attacks rather than regular misuse- or signature-based systems. By assigning contextual probabilities to network events, CBAM improves detection rates of low-volume remote access attacks and outperforms current state-of-the-art anomaly-based models in the detection of several attacks while retaining significantly lower false positive rates. Furthermore, CBAM retains low false positive rates for periods stretching several months. Our results provide good evidence that using contextual anomaly detection may in the future help decrease the threat of previously unseen vulnerabilities and malware aimed at acquiring unauthorised access on a host. We specifically focused on short-term anomalies as they are often an unavoidable by-product of an attack thus very difficult for an attacker to avoid without pre-existing control over the victim device or other network devices.

Chapter 6

Application to stepping-stone detection

6.1 Evading stepping-stone detection with enough chaff

Stepping-stones are used extensively by attackers to hide their identity and access restricted targets. Many methods have been proposed to detect stepping-stones and resist evasive behaviour, but so far no benchmark dataset exists to provide a fair comparison of detection rates. We propose a comprehensive framework to simulate realistic stepping-stone behaviour that includes effective evasion tools, and release a large dataset, which we use to evaluate detection rates for eight state-of-the-art methods. Our results show that detection results for several methods fall behind the claimed detection rates, even without the presence of evasion tactics. Furthermore, currently no method is capable to reliably detect stepping-stone when the attacker inserts suitable chaff perturbations, disproving several robustness claims and indicating that further improvements of existing detection models are necessary.

6.2 Introduction

The problem of stepping-stones detection (SSD) has been studied for over 20 years, yet the body of literature fails at providing an informative overview of the detection capabilities of current methods. In this paper, we set out to do just that by evaluating and comparing a number of selected state-of-the-art approaches on a new and independently generated dataset.

In a stepping-stone attack, malicious commands are relayed via a chain of compromised hosts, called stepping-stones, in order to access restricted resources and reduce the chance of being traced back. Real-world attacks using stepping-stone chains include Operation Aurora [?], Operation Night Dragon [?], the Black Energy [?] attack on the Ukrainian powergrid, and the MEDJACK [?] attack where medical devices were used as stepping-stones. The European Union Agency for Cybersecurity currently classifies stepping-stone attacks as one of the top ten threats to IoT-devices [?].

The detection of interactive stepping-stones is challenging due to various reasons. Attackers are not constrained to specific proxy techniques and can obfuscate relayed traffic with evasive tactics. Packet-based methods are computationally expensive while false-positives can render a method unusable. Like many intrusion attacks, stepping-stones are rare and there exist no public datasets, leading researchers to evaluate their methods on self-provided private data, which makes a direct comparison of the achieved results impossible.

In this work, we provide the following contributions:

1. We describe a framework to generate data that represents realistic stepping-stone data without bias to particular detection mechanisms. Our framework is scalable and capable of generating sufficient variety in terms of network settings and conducted activity.

2. We release a large and comprehensive dataset suitable for the training of machine-learning-based methods and in-depth performance evaluation. To our knowledge, this is the first public SSD dataset.

3. We re-implemented eight SSD methods that represent the current state-of-the-art and provide a fair evaluation of their capabilities in a number of settings.

4. Our evaluation shows that while most methods can accurately detect command propagation, detection rates plummet when appropriate chaff is inserted. This result disproves the claims made for multiple methods that their detection rates are robust against chaff perturbations.

The rest of the paper is organised as following: Section 6.2 provides an introduction and background to the problem of stepping-stone detection. Section 6.3 discusses the particular design of the data generation framework. Section 6.4 presents the dataset arrangement in terms of background and attack data and discusses evaluation methods. Section 6.5 discusses the selection process, properties, and implementation of the eight SSD methods that we implemented for evaluation. Section 6.6 discusses the results achieved by the implemented methods on the given data. Section 6.7 discusses related work.

6.2.1 Background

Stepping-stones were first conceptualised by Staniford-Chen and Heberlein in 1995 [?]. In an interactive stepping-stone attack, an attacker located at the origin host, called *host O*, sends commands to and awaits their response from a target, *host T*. The commands and responses are proxied via a chain of one or more intermediary stepping-stone hosts, called *host* S_1, ..., S_N, such as depicted in Fig. 6.1. Once a host S_i is brought under control, it can be turned into a stepping-stone with simple tools and steps. Some of the most common set-ups are port forwarding via SSH-tunnels, setting up a backpipe with NetCat, or using metasploit to set up a SOCKS proxy [?].

Stepping-stone detection (SSD) is a process of observing all incoming and outgoing connections on a particular host h_i and determining whether it is used to relay commands. This is generally done with no prior information about any other stepping-stone hosts $S_1, \ldots S_N$ or the endpoints O and T. A popular approach to

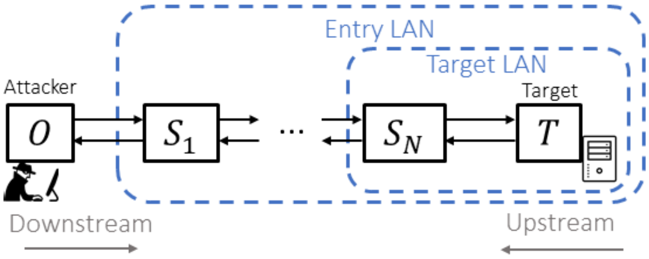


Figure 6.1: Depiction of an exemplary stepping-stone chain.

SSD is to compare connections pairwise to identify whether they carry the same information. To avoid detection, several evasive flow transformation techniques exist that aim at decreasing observable correlation between two connections in a chain.

- **Packet transfer delays/drops**: An attacker can choose to apply artificial delays to forwarded packets, or drop certain packets to cause retransmission, in order to create temporal disparity between connections. Researchers often assume the existence of a maximum tolerable delay [**?**].
- **Chaff perturbations**: Chaff packets do not contain meaningful content and are added to individual connections in a chain without being forwarded. Adding chaff perturbations can be used to shape the connection profile towards other traffic types.
- **Repacketisation**: Repacketisation is the practice of combining closely adjacent packets into a larger packet, splitting a packet into multiple smaller packets, or altering the packet content to change observed packet sizes and numbers.

In our evaluation, we set out to understand the effect of different evasive methods on detection rates.

6.3 Data generation setting

6.3.1 Containerisation

To ensure reproducibility, we rely on containerisation. A container is a standard unit of software that runs standalone in an isolated user space in order to remove platform dependencies and ensure repeatability. The use of containerisation for this project follows a traffic generation paradigm designed for machine learning, introduced by Clausen et al. [**?**].

6.3.2 Simulating stepping stones with SSH-tunnels and Docker

We want to capture data not only from one interaction in a fixed stepping-stone chain, but from many interactions and chains with different settings. For that, we run multiple simulations, with each simulation establishing a stepping-stone chain and controlling the interactions between host O and host T.

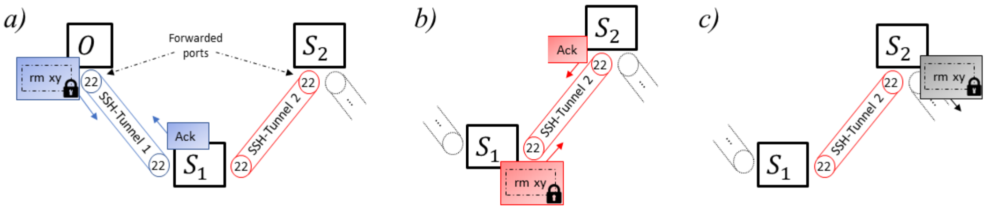


Figure 6.2: Depiction of the way a command is packetised, encrypted, and travels through the different stages of the stepping-stone chain via SSH-tunnels.

A simulation begins with the start-up of the necessary containers and ends with their takedown. We simulate host O, host T, and host $S_1, \ldots, S_n$ with SSH-daemon containers. To establish a connection chain, we connect these containers via SSH-tunnels, with the first tunnel forwarding a port from host O to host S_1, which is then forwarded to host S_2 by the second tunnel etc. As mentioned by Gordon Fraser [?], this is one of the most common pivoting methods for attackers. Traffic is captured both at host T and host S_n, which acts as the final stepping-stone in the chain. Fig. 6.2 depicts a packet transfer via an exemplary chain.

Simulating interactive SSH-traffic

In order to generate enough data instances representing interactive stepping stone behaviour, we automatised the communication between host O and host T. For each simulation, we generate a script which passes SSH-commands from host O to host T.

To mimic a user's actions, we compiled a command database which consists of common commands and their usage frequency, similar to [?].Commands are drawn randomly according to their usage frequency and concatenated to a script. Commands can either be atomic, such as "ls-la" or "pwd", or compound commands such as inputting text to a file. Command inputs are randomised appropriately when a compound command is drawn. A scripts ends once the *End*-command is drawn at random from the command catalogue.

To simulate human behaviour that is reacting to the response from host T, all commands are separated by *sleep*-commands for time t, which is drawn from a truncated Pareto-distribution. Paxson et al. [?] have shown that interpacket spacings corresponding to typing and "think time" pauses are well described by Pareto distributions with a shape parameter $\alpha \approx 1.0$.

Simulating different network settings

Hosts in a stepping-stone chains can be separated by varying distances. Some may sit in the same LAN, while others may communicate via the Internet from distant geographical locations, which influences the round-trip-time, bandwidth, and network reliability.

To retard the quality of the Docker network to realistic levels, we rely on the emulation tool NetEm, which allows users to artificially simulate network conditions such as high latency, low bandwidth, or packet corruption/drop [?]. We set the network settings and bandwidth limit for each host container individually before each simulation to allow hosts to experience different settings.

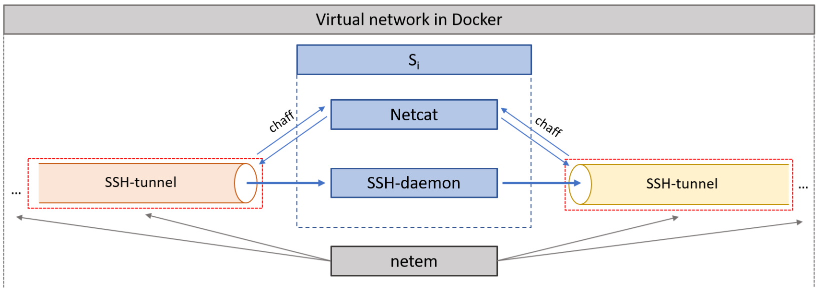


Figure 6.3: Depiction the simulation setup for each host in the chain.

6.3.3 Evasive tactics

Adding transfer delays

To simulate evasive behaviour, we add transfer delays to forwarded packets. This method, often called *jittering*, can destroy time-based watermarks in packet flows and help decrease observable correlation between two connections. The delays are added using NetEm. We draw delays from a uniform distribution, covering the interval $[0, \delta_D]$. This particular choice has been suggested by Padhye et al. [?] in order to mimic the interarrival distributions of streaming services. The value of δ_D is fixed before each simulation and can be varied to allow for different degrees of packet jittering. We explore values for δ_D up to 1500 ms, with values above leading to unstable communication. Results in Section 6.6 show that this is enough to render watermarking methods and most flow correlation methods obsolete.

Adding chaff perturbation

We insert chaff packets without actual information to individual connections in the chain using a Netcat client. To add and filter packets in a connection, we open additional ports in each SSH-tunnel that are however not forwarded through the entire chain. Padhye et al. [?] suggest to generate chaff that mimics the flow characteristics of streaming services to both spread the added perturbations evenly across the connection and increase the difficulty of detecting the perturbation itself. For this, packet sizes are drawn from a truncated Lognormal-distribution with mean μ_C, while transmission intervals are drawn from a uniform distribution that covers the interval $[\delta_C/2, \delta_C]$ to mimic a constant packet flow. By adjusting δ_C, we can control the amount of chaff sent.

Repacketisation

By design, SSH-tunnels perform repacketisation along with re-encryption and independent packet confirmations.

6.4 Evaluation data

We want to look at a variety of attack scenarios to highlight the strengths and weaknesses of different SSD approaches. We created three main attack datasets that contain different forms and amounts of evasive behaviour, and a smaller dataset to highlight the influence of different chain lengths.

To present a valuable false positive test, we provide three datasets with benign background traffic. The first contains general real-world traffic, while the second

	Label	Nr. of conn.	Purpose
Attack data	set BA	30.000	Baseline attack data without evasion tactics
	set DA	30.000	Inclusion of delays with varying δ_D
	set CA	30.000	Inclusion of chaff with varying δ_C
	set CL	40.000	Data from chains of different lengths, no evasion tacti
Background data	CAIDA	60.000	General background data
	SSH	20.000	Background data similar to attack commands
	Multim.	20.000	Background data similar to chaff perturbations

Table 6.1: Summary of different components in our evaluation data.

and third contain benign data that bears similar traffic characteristics as the generated attack data.

6.4.1 Stepping-stone data

We create our main datasets using a chain of four stepping-stones S_1, S_2, S_3, and S_4. We subdivide into three datasets: We first capture data without transfer delays and chaff perturbations in **dataset BA** (**b**aseline **a**ttack). We then capture data once with added transfer delays with varying δ_D to control delays in **dataset DA** (**d**elay **a**ttack), and once with added chaff perturbations of varying δ_C in **dataset CA** (**c**haff **a**ttack). Each dataset contains 30.000 connection pairs. We furthermore create a smaller **dataset CL** (**c**hain **l**ength) with differing numbers of stepping-stones (1,3,5, and 8 jumps) without transfer delays and chaff perturbations.

6.4.2 Benign data

We include real-world traffic traces, taken from the **CAIDA** *2018 Anonymized Internet Traces* dataset [**?**], as overall background traffic. This data contains traces collected from high-speed monitors on a commercial backbone link, and is often used for research on the characteristics of Internet traffic.

To sufficiently test for false-positive, we also need to include benign traffic that has similar characteristics to the attack traffic and was generated in a similar network environment. We created a set of interactive SSH-connections that communicate directly between the client and the server without a stepping-stone. We follow the same procedure as described in Section 6.3.2.

Since we generate perturbations with multimedia streams characteristics, we additionally want to test for false-positives against actual multimedia stream traffic. For that, we captured traffic from a Nginx-server streaming randomised video to a client.

We merge the three datasets to create our benign background dataset, with the CAIDA part containing 60.000 connection pairs, while the other two each contain 20.000 connection pairs. The amount of SSH traffic and multimedia streams in this setting is inflated from a realistic setting (up to 0.2% of flows for SSH and up 3% for video streaming [**?**]) to highlight the strengths and drawbacks of SSD methods, which we consider in the evaluation. In Section 6.8, we analyse false-positives for each dataset individually. Table 6.1 summarises the different parts in our evaluation data.

6.4.3 Evaluation methodology

To create a fair playing field for the selected SSD methods, we only look at connections that exchange more than 1500 packets and exclude shorter connections from both the data. The number of packets necessary for detection should ideally be a low possible to enable early detection. The chosen number of 1500 packets seems like a suitable minimal limit since all of the selected methods are designed to make successful detection with 300-1500 packets. Furthermore, there were no connections with less packets in the stepping-stone dataset. True stepping stone connections are rare compared to benign ones, making their detection an imbalanced classification problem. An appropriate evaluation measure for imbalanced data are false positive and false negative rates as well as the *Area-under-ROC-curve* (AUC) for threshold-based methods.

6.5 Selected SSD methods and Implementation

A range of underlying techniques exist for SSD, and we try to include approaches from every area to create an informative overview and highlight strengths and weaknesses. We surveyed publications to create a collection of SSD methods. We started with the publications from surveys [**?**, **?**], and then added impactful recent publications found via Google Scholar[1]. From here, we selected approaches based on the following criteria:

1. The achieved detection and false positive rates claimed by the authors,
2. and whether the model design shows robustness against any evasion tactics as claimed by the authors.
3. We always selected the latest versions if a method has been improved by the authors.

Table 6.2 contains a summary of the included methods. Especially for traditional packet-correlation as well as robust watermarking and anomaly-based methods, there has been little developments since the early 2010s. We labelled each method to make referring to it in the evaluation easier.

PContext, 2011

Yang et al. [**?**] compare sequences of interarrival times in connection pairs to detect potential stepping-stone behaviour. For that, the contextual distance of a packet is defined as the packet interarrival times around that packet. The authors focus on *Echo*-packets instead of *Send*-packets to resist evasion tactics.The authors evaluate their results with up to 100% chaff ratio with 100% detection rate.

[1] keywords “connection”, “correlation” “stepping-stone”, “detection”, “attack”, “chaff perturbation”

Category	Approach	TP	FP	Robustness	Label
Packet-corr.	Yang, 2011 [?]	100%	0%	jitter/< 80% chaff	PContext
Neural networks	Nasr, 2018 [?]	90%	0.0002%	small jitter	DeepCorr
	Wu, 2010 [?]	100%	0%	-	WuNeur
RTT-based	Yang, 2015 [?]	not provided		50% chaff	RWalk
	Huang, 2016 [?]	85%	5%	-	Crossover
Anomaly-based	Crescenzo, 2011 [?]	99%	1%	jitter/chaff	Ano1
	Huang, 2011 [?, ?]	95%	0%	> 25% chaff/ > 0.2s jitter	Ano2
Watermarking	Wang, 2011 [?]	100%	0.5%	< 1.4s jitter	WM

Table 6.2: Summary of included SSD-methods along with the claimed true positive and false positive rates and evasion robustness by the corresponding authors. We added labels to each method for later reference.

WuNeur, 2010

Wu et al. [?] propose a neural network model based on sequences of RTTs , which are fed into a feed-forward network to predict the downstream length of the chain. The network itself only contains one hidden layer and achieves good results only if RTTs are small, i.e. when the stepping-stone chain is completely contained within one LAN-network.

DeepCorr, 2018

Nasr et al. [?] train a deep convolutional neural network to identify connection correlation from the interarrival times and packet sizes in each connection. The trained network is large with over 200 input filters, and consists of three convolutional and three feed-forward layers. On stepping-stones, the authors achieve a 90% detection rate with 0.02% false positives.

RWalk, 2015

Yang et al. [?] combine packet-counting methods and RTT mining methods to improve detection results from [?]. THe model resists chaff perturbation by estimating the number of round-trips in a connection via packet-matching and clustering to determine if the connection is being relayed.

C-Over, 2016

Huang et al. [?] use the fact that in long connection chain, the round-trip-time of a packet may be longer than the intervals between two consecutive keystrokes. This will result in cross-overs between request and response, which causes the curve of sorted upstream RTTs to rise more steeply than in a regular connection.

Ano1, 2011

Crescenzo et al. [?] have proposed an anomaly-based methods to detect time delays and chaff perturbations in a selected connection. Packet time-delays are detected if RTTs exceed a threshold, while chaff detection compares the similarity of downstream with upstream sequences. The authors claim detection for chaff ratios 25% or more, and for delays introduced to up to 70% of all packets.

Ano2, 2011/2013

Huang et al. [?, ?] proposed an anomaly-based method to detect chaff and delay perturbations since interarrival times in regular connections tend to follow a Pareto or Lognormal distribution, which chaffed connections supposedly do

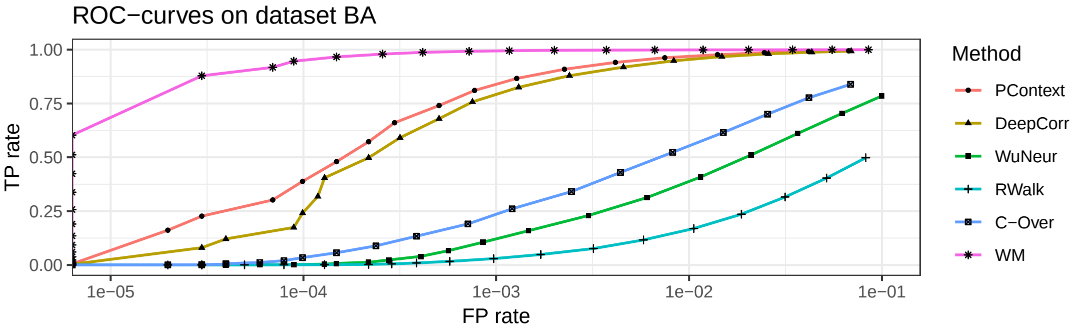


Figure 6.4: ROC-curves for different SSD methods on dataset BA (no evasive tactics). Anomaly-based methods are excluded.

	PContext	DeepCorr	WuNeur	RWalk	C-Over	WM
AUC	0.998	0.997	0.938	0.853	0.965	0.9998

Table 6.3: AUC-scores for different methods on stepping-stone data without evasive tactics.

not. The authors state 95% detection rate at 50% chaff ratio and more while retaining zero false positives using a small set of interactive SSH stepping-stone connections.

WM, 2010

Watermarking typically yields very low false-positives for connection correlation. Wang et al. [?] provide an approach that offers at least some resistance against timing perturbations. The authors assume some limits to an adversary's timing perturbations, such as a bound on the delays.The authors state 100% TP with 0.5% FP with resistance against timing perturbations of up to 1.4s.

6.6 Results

6.6.1 Data without evasion tactics

First, we look at the detection rates for traffic from stepping-stones that did not use any evasive tactics, i.e. $S_1, \ldots, S_4$ are only forwarding commands and responses. The successful detection of this activity with low false-positives should be the minimum requirement for any SSD method. Since anomaly-based approaches aim to only detect evasive behaviour, we exclude them from this analysis.

Fig. 6.4 depicts the calculated ROC-curves, which plot the true positive rate against the false positive rate for varying detection thresholds. Table 6.3 depicts the overall AUC-scores.

Unsurprisingly, the watermarking method achieves high detection results with very low false-positives. Both the PContext and DeepCorr models start to yield good detection results of around 80% at a FP rate lower than 0.1%, with the PContext method slightly outpacing the DeepCorr method. RTT-based methods seem to not perform as well compared to the other included methods. Overall, the observed ROC curves seem to be in agreement with the stated detection rates of the selected methods except for RWalk.

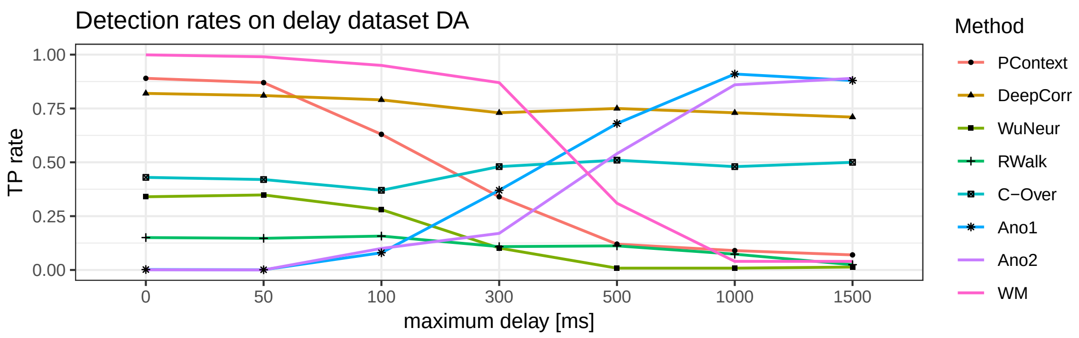


Figure 6.5: Detection rates in dependence of δ_D for different methods on dataset DA with a fixed FP rate of 0.4%.

	PContext	DeepCorr	WuNeur	RWalk	C-Over	Ano1	Ano2	WM
AUC	0.638	0.995	0.613	0.641	0.952	0.997	0.996	0.562

Table 6.4: AUC-scores for SSD methods with added transfer delays at $\delta_D = 1000ms$.

6.6.2 Delays

We now consider the effect of transfer delays added by the attacker to packets on the detection rates. For that, we pick detection thresholds for each SSD methods corresponding to a FP rate of 0.4% as most methods are able to achieve at least moderate detection results at this rate. We look at delays added to only to outgoing packets on S_4, the last stepping stone in the chain. Fig. 6.5 depicts evolution of detection rates in dependence of the maximum delay δ_D.

As visible, both anomaly-based methods are capable of detecting added delays relatively reliably above a certain threshold. Furthermore, both the detection rates of DeepCorr and the RTT-based C-Over only decrease slightly under the influence of delays. Detection rates for all other methods decrease significantly to the point where no meaningful predictions can be made. This is also reflected by the AUC-scores for traffic with $\delta_D = 1000ms$, given in Table 6.4.

While the WM method is robust against transfer delays up to $\delta_D = 500$ms, this value is smaller than the one claimed by the authors. This might however be a result of the slightly smaller quantisation step size that we used. It is surprising that the PContext method shows only little robustness against transfer delays, which contradicts the authors claims, potentially due to the incorrect assumption that relying on *Echo*-packets are not subject to transfer delays.

6.6.3 Chaff

We now consider the effect of chaff perturbations added by the attacker to individual connections on the detection rates. Again we pick detection thresholds for each SSD methods corresponding to a FP rate of 0.4%.

Chaff packets are added to both the connection between S_3 and S_4 as well as between S_4 and host T as described in Section 6.3.3. Fig. 6.6 depicts evolution of detection rates in dependence of the ratio of number of chaff packets to packets from the actual interaction.

As visible, all methods struggle to detect stepping stones once the chaff packets

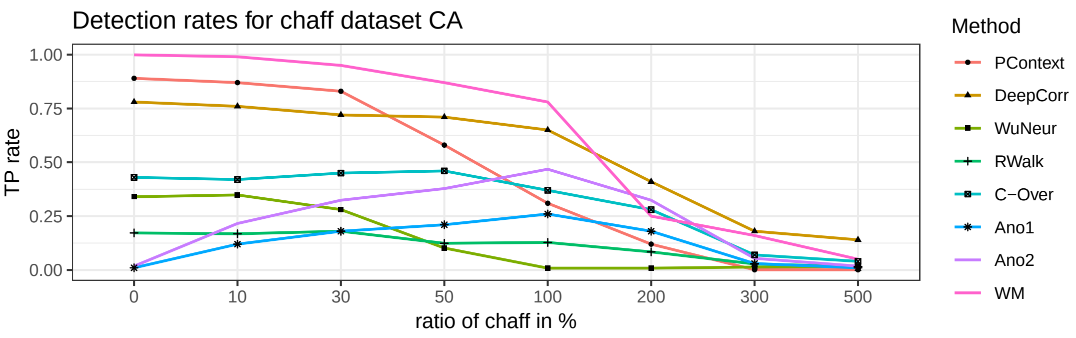


Figure 6.6: Detection rates in dependence of δ_C for different methods on dataset CA with a fixed FP rate of 0.4%

	PContext	DeepCorr	WuNeur	RWalk	C-Over	Ano1	Ano2	WM
AUC	0.639	0.886	0.615	0.641	0.589	0.782	0.738	0.839

Table 6.5: AUC-scores for SSD methods with added chaff at 300% ratio.

become the majority of the transferred traffic. This is also evident from the AUC-scores given in Table 6.5. Several approaches claimed to be resistent to chaff perturbations, however prior evaluations were limited chaff ratios below 100% without obvious reason.

It is surprising that the anomaly detection methods do not perform better at detecting chaff perturbations. Chaff in both approaches was however evaluated with different traffic generation distribution and not compared against a background of traffic following a similar generation distribution, which could explain the disagreement between the results we are finding here.

Overall, these results are in disagreement with the ”robustness” claims made for four of the selected approaches, namely PContext, RWalk, Ano1, and Ano2.

6.6.4 Summary

Overall, detection rates on dataset BA are mostly in line with the claimed capabilities except for RWalk, although detection rates are slightly lower than stated by most authors. Delay perturbation increases detection difficulty for most methods, except for Ano1, Ano2, and DeepCorr, which contradicts robustness claims for PContext and to some extend WM. Our inserted chaff perturbations however render detection impossible for all methods examined, which contradicts robustness claims for PContext, Ano1, Ano2, and RWalk, even though the claims were based on lower chaff levels.

As discussed in Section 6.9 and 6.10, longer chains yield higher detection rates for RTT-based methods while Different network transmission settings seem to have overall little influence on detection rates.

6.7 Related work

6.7.1 Testbeds and data

In 2006, Xin et al. [?] developed a standard test bed for stepping-stone detection, called *SST* that generates interactive SSH and TelNet connection chains with

	PContext	DeepCorr	WuNeur	RWalk	C-Over	Ano1	Ano2	WM
CAIDA	0.36	0.46	0.47	0.67	0.53	0.48	0.35	0.81
SSH	0.53	0.46	0.21	0.28	0.27	0.05	0.02	0.08
multimedia	0.11	0.08	0.32	0.04	0.20	0.47	0.63	0.11

Table 6.6: Relative contribution in % of different benign data to the FP rate.

variable host numbers. In contrast to our work, the authors give little detail on implemented evasive tactics, and is not available anymore.

An approach to use publicly available data comes from Houmansadr et al. [?],who simulate stepping stones by adding packet delays and drops retroactively to connections in the CAIDA data [?]. While this procedure seems sufficient for the evaluation of watermarking methods, it falls short on simulating the effects of an actual connection chain and leaves out chaff perturbations.

We find that when authors evaluate methods on self-generated data, tested evasive behaviours are often lacking analytical discussion and their implementations are too simplistic, leading to increased detection rates. An example of this can be seen in the evaluation of Ano1 [?], where a standard option in netcat is used to generate chaff perturbations for evaluation, or for PContext [?] where simulated chaff is added randomly after the traffic collection.Furthermore, often a relatively low limit on the amount of inserted chaff perturbations is assumed without obvious reason, thus avoiding evaluation at higher ratios.

6.8 False positives

Table 6.6 depicts the relative contribution[2] at $FP = 0.4\%$ of each of the three benign data types to the overall false positive rate. Most methods have more problems with the heterogeneous nature the CAIDA traces, with only PContext and DeepCorr seeing most false positives in the SSH traffic.

The multimedia traffic is causing most problems for the anomaly-based methods, persumably because it follows a similar distribution as the generated chaff perturbations.

6.9 Influence of chain length

In this section, we look at the effect of differing chain lengths on the detection rates. We only focus on RTT-based methods here since the other methods should and do not see a significant effect from varying chain lengths[3]. Since RTT-based methods aim to measure the effect of packets travelling via multiple hosts, it is unsurprising that they perform better at detecting longer chains.

Of the RTT-based methods, only C-Over was able to yield consistent detection rates under transfer delays. Interestingly, if the C-Over method is applied to connections between S_3 and S_4 instead of between S_4 and the target, detection rates decrease in the same manner as for other RTT-based methods. This is not

[2] after adjusting for their weight

[3] For non-RTT-methods, the detection rate error (2.6% – 6.5%) for each length was larger than the detection rate differences (0.2% – 3.7%) across different lengths.

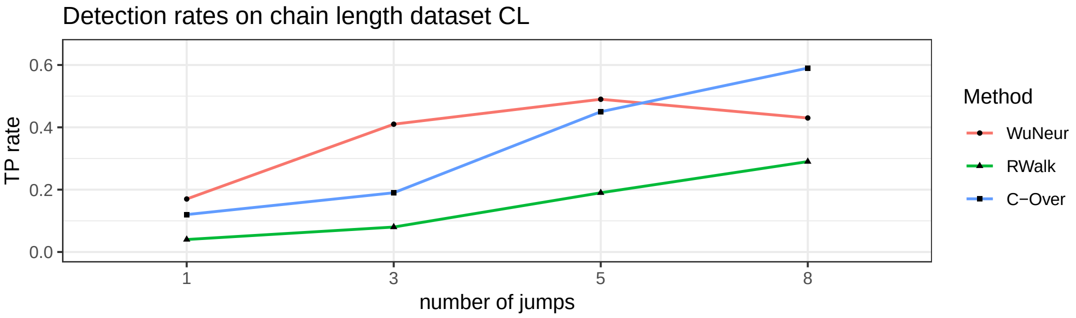


Figure 6.7: Detection rates in dependence of chain length for different methods on dataset CL with a fixed FP rate of 0.4%

	Value	TP deviation from average				
		DeepCorr	WuNeur	RWalk	C-Over	WM
RTT	5ms	−0.2%	+41.3%	−42.3%	−36%	+0.03%
	70ms	−5.6%	−5.8%	+35.1%	+51%	−2.2%
Packet loss	0%	+1.2%	+1.3%	+2.1%	+4.3%	+0.02%
	7%	−9.1%	−1.1%	−3.1%	−7.3%	−9.7%

Table 6.7: Influence of network congestion on detection rates at a fixed FP rate of 0.4%. The given percentages are describing the change of the detection rate under the given congestion setting when compared to the overall average.

surprising as the underlying assumption for robustness for this approach relies on Echo-packets not being delayed.

6.10 Influence of network settings

Finally, we look at the effect of different nework settings. We only show methods that show significant effects and omitted bandwidth from the evaluation as different values do not seem to have any effect on detection rates[4].

As visible in Table 6.7, the three RTT-based methods show different responses to small/large average round-trip-times. While WuNeur, as expected from prior results, performs better in LAN settings, detection rates of the RWalk and C-Over methods are boosted by larger RTTs. All methods profit from lower packet losses.

6.11 Conclusion

In this work, we set out to evaluate the state-of-the-art of SSD methods using a comprehensive data generation framework. Our framework simulates realistic stepping-stone behaviour with SSH-tunnels in different settings and varying amounts of evasive perturbation tactics. We will release a large dataset that highlights multiple aspects in SSD, and is suitable to train ML-based methods.

[4] For all methods, the detection rate differences (0.7%−6.2%) were smaller across bandwidths than the overall detection rate errors (2.6% − 6.5%).

Overall, our results show that attackers can reliably evade detection by using the right type and amount of chaff perturbation, which disproves several claims made about the robustness against this evasive tactic. Although to a lesser degree, our implemented delay perturbations still affect detection rates for most methods.

Currently, it seems that watermarking methods are most suited to reliably detect simple stepping-stones in real-life deployment. The performance of DeepCorr indicates that deep neural networks show the most potential at detecting attacks that use chaff or delay perturbations if they are trained on suitable data. We find that detection and false-positive rates for RTT-based methods are significantly lower than for other methods.

Chapter 7

Requirements for Machine Learning

7.1 Traffic Generation using Containerization for Machine Learning

The design and evaluation of data-driven network intrusion detection methods are currently held back by a lack of adequate data, both in terms of benign and attack traffic. Existing datasets are mostly gathered in isolated lab environments containing virtual machines, to both offer more control over the computer interactions and prevent any malicious code from escaping. This procedure however leads to datasets that lack four core properties: heterogeneity, ground truth traffic labels, large data size, and contemporary content. Here, we present a novel data generation framework based on Docker containers that addresses these problems systematically. For this, we arrange suitable containers into relevant traffic communication scenarios and subscenarios, which are subject to appropriate input randomization as well as WAN emulation. By relying on process isolation through containerization, we can match traffic events with individual processes, and achieve scalability and modularity of individual traffic scenarios. We perform two experiments to assess the reproducability and traffic properties of our framework, and demonstrate the usefulness of our framework on a traffic classification example.

7.2 Introduction

The exponential growth of data availability enabled the machine learning revolution of this decade that transformed many areas of our lives. Ironically, security oriented data describing computer networks is notoriously hard to obtain, and researchers struggle to evaluate new network intrusion detection systems (NIDS) or similar tools on suitable network traffic data. Well-designed datasets are such a rarity that researchers often rely on datasets that are well over a decade old [**?**, **?**], calling into question their effectiveness on modern traffic and attacks. The lack of quantity, variability, meaningful labels, and ground truth has so far slowed scientific progress and objective and appropriate measurements on ML-based network security methods.

Privacy and security concerns discourage network administrators to release rich and realistic datasets for the public. Network traffic produced by individuals contains a mass of sensitive, personal information, such as passwords, email addresses, or usage habits, requiring researchers to expend effort anonymising the dataset [?]. To examine malicious behaviour, researchers are often forced to build artificial datasets using isolated machines in a laboratory setting to avoid damaging operational devices. Background traffic is usually generated in real-time from scripts executed on the virtual machine, which constrains both the amount and heterogeneity of the data. Current experiments show that it is possible to use large network of virtual machines for traffic generation [?, ?], however we did not find any publicly available dataset of that category.

Existing network intrusion detection datasets are predominantly designed to support a broad range of applications, and are collected in a static manner, unable to be modified or expanded. This proves to be a serious defect as the ecosystem of intrusions is continually evolving. Furthermore, it prohibits a more detailed analysis of specific areas of network traffic due to the available data only being a fraction of the original dataset. To combat this, new datasets must be periodically built from scratch.

Allowing researchers to create datasets dynamically to circumvent these issues would be extremely beneficial. We propose a such framework, based on application containers using Docker [?]. Docker is a service for developing and monitoring containers, also known as OS-level virtual machines. By moving from virtual machines to containers, we enable the scalable, modular, and dynamic creation of network traffic datasets. Since Docker containers can be arranged in complex settings with a few commands, it is a lot easier with containers to script a variety of network activities thus increase the heterogeneity and realism of the generated data.

Furthermore, each Docker container is highly specialized in its purpose, generating traffic related to only a single application process. Therefore, by scripting a variety of Docker-based *scenarios* that simulate benign or malicious behaviours and collecting the resultant traffic, we can build a dataset with perfect ground truth, something that has so far not been possible for network traffic.

Finally, the most import reason to rely on Docker for containerization is that many containerized applications are shared on the Docker Hub platform.

This work provides the following contributions:

1. We present a novel network traffic generation framework that is designed to improve several shortcomings of current datasets for NIDS evaluation. This framework is openly accessible for researchers and allows for straightforward customization.

2. We define four new requirements a network intrusion dataset should fulfil in order to be suitable to train machine-learning based intrusion detection methods.

3. We perform a number of experiments to demonstrate the suitability and utility of our framework.

7.2.1 Outline

The remainder of the paper is organized as follows. Section 7.3 discusses existing NIDS datasets and the problems that arise during their usage as well as background information about network traffic data formats and virtualization methods. The section concludes with a set of requirements we propose to improve the training and evaluation of machine-learning-based methods. Section 7.4 describes the general design of our framework, and how it improves on the discussed problems in existing datasets. We also discuss a specific example in detail. Section 7.5 discusses several experiments to validate the improvements and utility our framework provides. Section 7.6 concludes the results and discusses limitations of our work and directions for future work.

7.3 Background

7.3.1 Data formats

Computers in a network communicate by sending *network packets* to each other, which are split into the control information, also called packet header, and the user information, called payload. The payload of a packet in general carries the information on behalf of an application and can be encrypted, while the header contains the necessary information for the correct transmission of the packet, including the transmission protocol layer, IP addresses, etc. Methods using packet-level input can be divided into payload inspection, header-based, or hybrid. Packets are usually stored in the widespread *pcap* format.

The majority of packets are exchanged between two hosts within bidirectional *connections*. Another common format of network traffic information is based on connection summaries, also called *network flows*. RFC 3697 [**?**] defines a network flow as a sequence of packets that share the same source and destination IP address, IP protocol, and for TCP and UDP connections the same source and destination port. A network flow is usually represented by this information along with additional information such as the start and duration of the connection as well as the total number of packets and bytes transferred.

7.3.2 Related work and existing datasets

To evaluate their ability to model the behaviour of a network and to identify malicious activity and network intrusions, new methods have to be tested using existing datasets of network traffic. This network should ideally contain realistic and representative benign network traffic as well as a variety of different network intrusions. However, as network traffic contains a vast amount of information about a network and its users, it is notoriously difficult to release a comprehensive dataset without infringing the privacy rights of the network users [**?**]. Furthermore, the identification of malicious traffic in network traces is not straightforward and often requires a significant amount of manual labelling work. Introducing malicious software into an enterprise network would be impossible for ethical and security reasons. For that reason, only about four structured datasets for network intrusion containing real-world traffic and attacks are available openly. The most recent and notable real-world datasets have been released from the Los Alamos National Laboratory (LANL) in 2015 and 2017 [?, ?], and the University of Granada (UGR) in 2016 [?]. Both datasets contain network flow traffic data from a large number of hosts collected over multiple months, giving an accurate representation of medium- to large-scale structures in benign traffic. However, the amount of attack data is small and insufficient for accurate detection rate estimation. Furthermore, packet-level data is not available for both datasets. Other real-world datasets, such as CAIDA 2016 [?] or MAWI 2000 [?], provide packet headers, but are unstructured and contain no labeled attack data at all.

To improve the lack of attack traffic in NIDS datasets, several artificially created datasets have been proposed. For this, a testbed of virtual machines is usually hosted in an enclosed environment to prevent any malicious code from spreading to other machines on other networks. To generate attack traffic, these machines are then subject to a selection of attack carried out by other machines in the environment. Benign traffic is generated using commercial traffic generators such as the *IXIA PerfectStorm tool*, or by scripting a selection of tasks for each machine. Synthetic datasets cover a smaller timeframe and contain traffic from a small number of hosts. Notable examples are the CIC-IDS 2017 dataset from the Canadian Institute for Cybersecurity [?], and the UNSW-NB 2015 dataset from the University of New South Wales [?]. Both datasets contain traffic from a variety of attacks, and are available as packet headers or as network flows with additional features crafted for machine-learning. While the benign traffic for the CIC-IDS 2017 data was generated using scripted tasks from a number of host profiles, the benign data for the UNSW-NB 2015 data is a mixture of captured real traffic from another subnet and traffic generated using a commercial traffic replicator.

We omitted the synthetic KDD-Cup 1999 and the DARPA 1998 datasets along with their derivates from the discussion as they are well-known to be outdated and contain unrealistic benign traffic, artificially high benign/attack data ratios, and artifacts stemming from communication simulations [?, ?], problems which have been addressed by most modern datasets.

Container networks have recently been adopted to conduct traffic generation experiments, such by Fujdiak et al. [?] who use containerized web servers to collect DoS-traffic. Furthermore, significant effort has been put into the creation of large-scale virtualization frameworks to provide automatized network testbeds [?, ?].

7.3.3 Problems in modern datasets

The difficulty of obtaining malicious traffic in real-world captures means that the performance of new network intrusion detection algorithms are almost exclusively evaluated on synthetic datasets. Potential disadvantages of synthetic compared to real-world datasets have been discussed by several authors [?, ?]. However, none address problems in the particular design of such synthetic testbeds that are holding machine-learning based methods back in performance and from getting more widespread application. Here, we focused on this aspect and four design problems common among modern synthetic datasets.

Lack of variation To generate benign traffic, a selection of activities is scripted and executed on virtual machines. Activities are selected to cover the most prominent protocols, but seldom to cover the range of subactivites that each protocol offers. Instead, the manner in which each protocol is used is highly-restricted, and there are doubts about whether this traffic is representative of its real-world equivalent usage [**?**]. An illustrative example of the restricted protocol activity in synthetic datasets can be seen in the CIC-IDS 2017 dataset. Here, the vast majority of successful FTP transfers consist of a client downloading a single text file containing the Wikipedia page for 'Encryption' several hundred times in a day. In reality, FTP is used for a large number of tasks, which can occur in random order with varying input sizes and parameters.

In addition to that, implemented test bed environments are usually separated from external influence or even virtualized, which isolates them from fluctuations and faults introduced by the complexity of modern networking. These include packet delays through network congestions, unexpected connection drops or resets, and out-of-order arrivals, all of which lead to variations in the response behavior of particular services.

This general lack of variation in individual protocols leads to observed homogeneity both on a packet exchange level and on a network flow level, and thus to clearer structures in the data. Identifying separations of malicious and benign activity or between different services consequently becomes easier, which leads to overoptimistic results in the evaluation of machine-learning based methods. It is therefore clear that traffic variation is a crucial aspect of a comprehensive intrusion detection dataset. parameter

Lack of ground truth To evaluate machine-learning-based methods that distinguish between different types of network traffic data, we need to verify that separating structures in the model correspond to distinct computational actions, using traffic labels. Most obvious is the labelling of benign and malicious traffic. More granular labels are desirable to distinguish between several different types of network traffic. An example for this is the design of 'stepping stone' detection methods, where researchers try to detect connections relayed over a jump-host. Similarity or correlation metrics that measure the closeness of two connections are a popular tool. To understand such a measure, ground truth about how computationally similar two connections are and what type of behavior they represent is necessary. Other areas that look at small-scale traffic structures and would benefit from detailed traffic labels include protocol verification, traffic classification, traffic disaggregation, or exploit discovery.

Ground truth labels for network traffic are hard to obtain. The network traffic produced by a typical PC will invariably contain traffic originating from background processes, such as software updates, authentication traffic, network discovery services, advertising features, as well as many other sources. To separate traffic from different origins retrospectively is often hard, if not impossible. Source attribution through port numbers is unreliable because port numbers can be dynamically allocated and are not restricted to particular processes, and processes can open connections on multiple ports at the same time. All of these reasons mean that the identification of different computational operations from captured traffic is often infeasible. Therefore, no public NID dataset currently considers the inclusion of ground truth traffic labels.

Static design A released dataset can only contain data that is representative a system at the time of creation. In contrast to other many other data sources for machine-learning, network traffic, both benign and malicious, is constantly changing as computational protocols and systems evolve. All available NIDS datasets today have been created in a static manner , so that a fixed test bed of host machines is created designed to contain specific vulnerabilities to the selected attacks. This makes it very hard to change the test bed and thus adjust the dataset to updated traffic structures. Allix et al. [?] claim that it is impossible to release a NID dataset that is truly representative of the real-world attacks due to the inherent secrecy of the intrusion ecosystem and the rate at which it develops.

Limited size Today's machine-learning revolution was supercharged by the exponential growth of available data. Larger amounts of data mean that a given model can identify more complex structures that remain invariant in noisy environments and thus generalize better. Although the amount of globally transmitted network traffic is growing every year, the size of available NIDS datasets is limited by small host numbers, typically 5-10, and short capture periods, at maximum a 5-6 weeks, inherent to test bed captures. This means that traffic models can experience difficulties to generalize over specific traffic types which represent a smaller fraction of the total dataset. In an ideal setting, researchers would have the ability to generate arbitrary amounts of specific traffic types.

7.3.4 Containerization with Docker

Virtual machines (VMs) share the same hardware infrastructure as the host machine. VMs necessitate the use of hypervisors, software responsible for sharing the host OS's hardware resources, such as memory, storage and networking capabilities. OS-level virtualization, also known as *containerization*, is a virtualization paradigm that has become popular in recent years due to its lightweight nature and speed of deployment. In contrast with standard VMs, containers forego a hypervisor and the shared resources are instead kernel artifacts, which can be shared simultaneously across several containers. Although this prevents the host environment from running different operating systems, containerization incurs minimal CPU, memory, and networking overhead whilst maintaining a great deal of isolation [?].

The main advantage of using containers for traffic generation is the isolation of individual applications. This enables us to gather ground truth about the traffic origin, and enables us to easily extend, modify, and scale our traffic generation framework, which would not be possible when relying on VMs.

Docker container *Docker* is a software platform that allows for the creation, maintenance and deployment of containers. In Docker's terminology, a container is a single, running instance of a Docker *image*. Docker images are defined via a text file known as the `Dockerfile`, which consists a series of commands that modify an underlying *base image*, usually a containerized OS. Example commands

```
FROM ubuntu
MAINTAINER XYZ (email@domain.com)
RUN apt−get update
RUN apt−get install −y nginx
ENTRYPOINT [”/usr/sbin/nginx”,”−g”,”daemon off;”]
EXPOSE 80
```

Figure 7.1: Example of `Dockerfile` creating a nginx-container.

include installing libraries and copying files. Figure 7.1 displays a simple example of `Dockerfile`.

After each command is executed, the intermediate, read-only image is saved as a *layer*. These layers can be shared between containers. When a Docker image is run as a container, a final read-write layer is added and when the container is later stopped, this layer is discarded, preserving the integrity of the underlying layers. This allows Docker containers to be run repeatedly whilst always starting from an identical state.

Individual Docker containers are intended to be highly specialized in their purpose with each container running only a specific piece of software or application. Commonly used base images — such as Alpine Linux — have minimal background processes running during a container's lifetime. This means that the network traces of a Docker container can be associated with a specific application. The one-to-one correlation between containers and network traces allows us to produce labeled datasets with fully granular ground truths.

The Docker software platform includes a cloud-based repository called the Docker Hub [?] which allows users to download and build open source images on their local computers. At the time of writing, nearly 2.5 million images are available from the Docker Hub. Some common software — such as popular web-servers and databases — have officially maintained images. We use these as far as possible to simplify the production of our scenarios, and keep them close to software configuration used in practice.

Docker Networking Docker allows the creation of virtualized networks with one or more subnetworks, to which containers can connect via a virtualized network bridge. Containers attached to the bridge network are assigned an IP address and are able to communicate with other containers on their subnetwork. Containers can furthermore be connected to a host network, which allows communication with external networks using NAT via the host interface.

To host containers in an isolated network, we can create our own user-defined bridge networks, which provides greater isolation between containers [?]. Furthermore, this allows us to fix the subnet and gateway for our networks as well as the IP addresses of our containers, which simplifies scripting scenarios. Docker allows containers to share the same network interface. This enables us to assign the same IP address to multiple containers.

Netem The Docker engine also provides network access to Linux traffic control facilities such as *NetEm*, on which we will rely in this project. NetEm is a Linux toolkit for testing protocols by emulating properties of wide area networks [?].

```
version: '3'
services:
 webserver:
 image: nginx:alpine
 ports:
 - "80:80"
 networks:
 - app-network
 db:
 image: mysql:5.7.22
 ports:
 - "3306:3306"
 environment:
 MYSQL_DATABASE: laravel
 MYSQL_ROOT_PASSWORD: your_mysql_root_password
 networks:
 - app-network
networks:
 app-network:
 driver: bridge
```

Figure 7.2: Example of `Docker-compose` file launching a nginx- and a mysql-container in an isolated network.

It allows the user emulates variable delay, loss, duplication and re-ordering of packets on particular network interfaces.

Docker Compose Applications built using the Docker framework often need more than one container to operate, for example an Apache server and a MySQL server running in separate containers. We must build and deploy several interconnected containers simultaneously. Docker provides this functionality via `Docker compose`, a tool that allows users to define the services of multiple containers as well as the properties of virtual networks in a YAML file. By default, this file is named *docker-compose.yml*. This allows for numerous containers to be started, stopped and rebuilt with a single command in a consistent manner. This is particularly significant for our purposes; with `Docker compose`, we can launch several containers in a specific order, with a specific network configuration, whilst running specific commands within each container on start up. This ensures that our interactions are deterministic, barring any added randomization. Figure 7.2 displays an example of a simple `Docker compose` file.

7.3.5 Dataset Requirements

The primary task of this project is to provide a suite of Docker container compositions that is capable of generating traffic datasets suitable for machine-learning-based intrusion detection systems. This container suite is designed to address the criticism of current NIDS datasets discussed in Section 7.3.3. For this, we created a set of requirements that a modern intrusion detection dataset has to fulfil to address the problems discussed in Section 7.3.3:

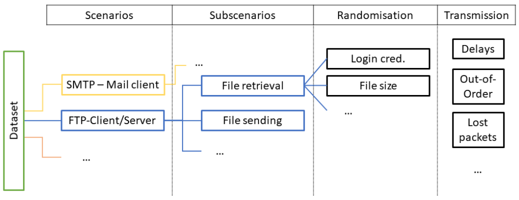


Figure 7.3: Visualization of the different levels at which traffic variation is introduced in DetGen.

Variation To ensure that we produce representative data for modelling, we want the traffic generated by our container suite to cover a sufficient number of protocols that are commonly found in real-world traffic and existing datasets. For malicious traffic, we want to ensure that the attacks are modern and varied, both in purpose and in network footprint. For each protocol, we want to establish several capture scenarios to encompass the breadth of that protocol's possible network traces. Communication between containers should be subject to the same disturbances and delays as in a real-world setting.

Ground truth Since ground truth is a main focus of this work, we want a capture scenarios to be consistent and reproducible in the traffic they generate. This way, we can be certain that a particular traffic trace corresponds to the capture scenario it was generated by, and can thus relate individual traffic events to computational operations. We discuss what it means for a scenario to be reproducible in detail in Section 7.5.1.

Modularity Traffic capture scenarios should be implemented in a modular way allow for a straightforward addition or modification of traffic capture modules without disrupting the rest of the container suite. This reduces the effort to adjust a dataset to changing traffic patterns and allows the addition of modern attacks traffic.

Scalability Each capture scenario should be running in a scalable manner to allow generation of large data quantities.

7.4 Design

To cover a range of activities, the containers in our framework are arranged in different configurations corresponding to particular *capture scenarios*. Running a given capture scenario triggers the launch of several Docker containers, each with a scripted task specific to that capture scenario. A simple exemplary capture scenario may consist of a containerized client pinging a containerized server. We ensure that each Docker container involved in producing or receiving traffic will be partnered with a `tcpdump` container, allowing us to collect the resulting network traffic from each container's perspective automatically.

We outline different stages within the creation of a dataset at which traffic variation is introduced. Figure 7.3 visualizes this process.

7.4.1 Scenarios

We define a *scenario* as a series of Docker containers interacting with one another whereby all resulting network traffic is captured from each container's perspective. This constructs network datasets with total interaction capture, as described by Shiravi et al. [?]. Each scenario produces traffic from either a protocol, application or a series thereof. Both benign and malicious activities are implemented as scenarios. Examples may include an FTP interaction, a music streaming application and client, an online login form paired with an SQL database, or a C&C server communicating with an open backdoor. A full list of currently implemented scenarios can be found in Section 7.4.9.

Each scenario is designed to be easily started via a single script that allows the user to set the length of the capture time, and the specification of particular subscenarios, discussed below. Scenarios can be repeated indefinitely without further instructions and be run in parallel, therefore allowing the generation of large amounts of data.

Our framework is modular, so that individual scenarios are configured, stored, and launched independently. Adding or reconfiguring a scenario has no effect on the remaining framework.

7.4.2 Subscenarios

In contrast to scenarios, *subscenarios* provide a finer grain of control over the traffic to be generated, allowing the user to specify the manner in which a scenario should develop. The aim of having multiple subscenarios for each scenario is to explore the full breadth of a protocol or application's possible traffic behavior. For instance, the SSH protocol can be used to access the servers console, to retrieve or send files, or for port forwarding, all of which may or may not be successful. It is therefore appropriate to script multiple subscenarios that cover this range of tasks.

The same applies to malicious activity. For instance, it would be naive for an SSH password bruteforcing scenario to always successfully guess a user's password. Instead, we include a second subscenario in which the password bruteforcer fails.

Subscenarios are specific to particular scenarios and can be specified when launching that scenario.

7.4.3 Randomization within Subscenarios

Scripting activities that are otherwise conducted by human operators often leads to a loss of random variation that is normally inherent to the activity. As mentioned in Section 7.3.3, the majority of successful FTP transfers in the CIC-IDS 2017 data consist of a client downloading a single text file. In reality, file sizes, log-in credentials, and many other variables included in an activity are more or less drawn randomly, which naturally influences traffic quantities such as packet sizes or numbers.

To account for these fluctuations, we identify variable input parameters within scenarios and their subscenarios and systematically draw them randomly from a suitable distribution. Passwords and usernames, for instance, are generated as a random sequence of letters with a length drawn from a Cauchy distribution, before they are passed to the corresponding container. Files to be transmitted are selected at random from a larger set of files, covering different sizes and file names.

7.4.4 Network transmission

Docker communication takes place over virtual bridge networks, so the throughput is far higher and more reliable than in real-world networks, with the Docker virtual network achieving a bandwidth of over 90 Gbits/s when measured using iPerf [?]. This level of speed and consistency is worrying for our purposes as packet timings will be largely identical on repeated runs of a scenario and any collected data could be overly homogeneous.

To retard the quality of the Docker network to realistic levels, we rely on emulation tools. As discussed in section 7.3.4, Netem is a Linux command line tool that allows users to artificially simulate network conditions such as high latency, low bandwidth or packet corruption in a flexible manner.

Although it is relatively straightforward to apply Netem commands to a Docker Bridge network, we decided not to invoke Netem in this manner as this would cause all network settings of all containers to be identical, such as all containers in a scenario having a latency of 50ms. Instead, we developed a wrapping script that applies Netem commands to the network interface of a given container, providing us with the flexibility to set each container's network settings uniquely. This script randomizes the values of each parameter, such as packet drop rate, bandwidth limit, latency, ensuring that every run of a scenario has some degree of network randomization if desired.

7.4.5 Capture

To capture traffic, we use containers running `tcpdump`, a widespread and free packet analyzer software that can capture packets arriving at or leaving from a network interface [?]. We attach `tcpdump` containers on every interface in the virtualized docker network and write packets into separate capture files. This allows us to capture traffic from the perspective of every container in a scenario, giving a complete view. In Section 7.4.8, we discuss how the collected capture files are coalesced into one dataset.

7.4.6 Implementation Process

The implementation process for each scenario follows broadly the same outline:

1. Select containers which provide the required services and identify the *primary* container/s for a given scenario which is/are dictating the container interaction. Then create and build a Dockerfile containing all necessary dependencies.

2. Identify different ways to use the service of the given scenario and define them into a set of subscenarios.

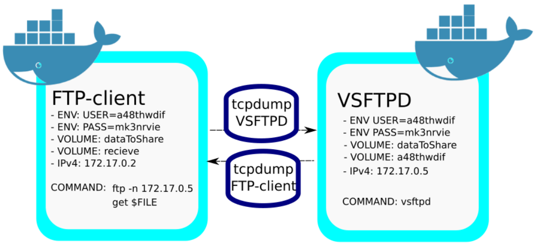


Figure 7.4: Diagram of FTP scenario

3. Design and implement the behavior for secondary containers to provide the required service to the primary container(s).

4. For each subscenario, identify variable input values and their appropriate range; then systematically implement their generation from appropriate distributions covering this range.

5. Add `tcpdump` containers to every network interface.

6. Create a `Docker compose` file that launches all containers simultaneously.

7. Finally, write a script that, upon running, calls this `Docker compose` file, applies a network emulation script to each container network interface, and allows the user to specify how long and how many times a scenario should be run.

Following the Docker guidelines [?], each container in our framework consists of a single service with a specialized purpose, with as few additional dependencies as possible. Moreover, we ensure that there are minimal inter-dependencies between the containers of a scenario. This allows us to easily modify and update containers as new versions of the underlying software are released.

7.4.7 Simple Example Scenario - FTP server

We review the design of a prototypical capture scenario, namely, an FTP server and client interaction. The interaction is initiated by a single script, which allows the user to specify the length of the interaction, the number of times the interaction takes place as well as the specific subscenario. The script generates a random ftp username and password, creating the necessary *User* directory on the host machine before calling the Docker-compose file which creates a bridge network. Subsequently, the necessary containers are then started which, in this case, consist of a VSFTPD server, a client with ftp instaled and two containers running tcpdump to capture all of the traffic emitted and received by the client and server respectively into separate `.pcap`-files. These `.pcap`-files are shared with the host machine via a shared volume. The host machine also shares:

- A *dataToShare* volume containing files that can be downloaded by the client.

- The *User* directory with the server, which contains the same files as the *dataToShare* folder.

- An empty *receive* folder with the client into which the files will be downloaded.
- The random username and password is shared with the client container so it can authenticate itself to the server.

Up to this point, no network traffic has been generated and the containers are now ready to begin communicating with one another. For this particular interaction between an FTP server and client, we want to ensure that it is possible to capture the many ways in which an FTP session may develop. For instance, the client may seek to download files via the `get` command or the `put` command, alongside many other possibilities. We define 13 possible capture subscenarios intended to encapsulate a wide range of potential FTP sessions. These include downloading a single file using `get` or `put`, downloading every file using `mget` or `mput`, deleting every file from the server and requesting files from the server without the necessary authentication.

After the scenario ends, both the *User* directory and any downloaded files are removed from the host machine. The containers are then stopped and the bridge network is torn down. All necessary containers, volumes and scripts are in the same position prior to initiating the scenario — barring any generated `.pcap`-files — allowing for the scenarios to be started repeatedly with minimal human interaction. The `.pcap`-files are tagged with information about the time of creation, executed scenario and subscenario, and the container generating the traffic.

7.4.8 Dataset creation

Our framework generates network datasets consisting of a single interaction, but it is possible to coalesce these datasets to create larger datasets with a wide variety of traffic, albeit with some caveats. Due to the networking constraints of the Docker virtual network, such as limitations regarding clashing ports, running many of our Docker scenarios simultaneously over a large period of time is unfeasible. Thus, to ensure that the generated traffic is suitably heterogeneous, numerous datasets must be generated before being coalesced into a main dataset. If done naively, this presents a problem. As discussed by Shiravi et al. [?], merging distinct network data in an overlapping manner can introduce inconsistencies. For instance, if one wanted to create a dataset containing both normal webserver traffic and traffic originating from a Denial of Service attack, it would not work to generate these two datasets separately before merging them together. If these two events really did occur simultaneously, the high network throughput of the latter would likely effect the packet timings of the former.

To avoid such inconsistencies, we create larger datasets by collecting data in consecutive chunks of fixed time. Within each chunk, several scenarios are run simultaneously. All `.pcap`-files collected during a given chunk can be merged together. It is then simple to stitch together all of these chunks into a single `.pcap`-file using a combination of `Mergecap` [?] and `Editcap` [?]. This allows us to shift the timings of each `.pcap`-file by a fixed amount such that all of our chunks occur in succession whilst maintaining the internal consistency of each chunk.

7.4.9 Scenarios

Our framework contains 29 scenarios, each simulating a different benign or malicious interaction. The protocols underlying benign scenarios were chosen based on their prevalence in existing network traffic datasets.These datasets consist of common internet protocols such as HTTP, SSL, DNS, and SSH. According to our evaluation, our scenarios can generate datasets containing the protocols that make up at least 87.8% (MAWI), 98.3% (CIC-IDS 2017), 65.6% (UNSW NB15), and 94.5% (ISCX Botnet) of network flows in the respective dataset. Our evaluation shows that some protocols that make up a substantial amount of real-world traffic are glaringly omitted by current synthetic datasets, such as BitTorrent or video streaming protocols, which we decided to include.

In total, we produced 17 benign scenarios, each related to a specific protocol or application. Further scenarios can be added in the future, and we do not claim that the current list exhaustive. Most of these benign scenarios also contain many subscenarios where applicable.

The remaining 12 scenarios generate traffic caused by malicious behavior. These scenarios cover a wide variety of major attack classes including DoS, Botnet, Bruteforcing, Data Exfiltration, Web Attacks, Remote Code Execution, Stepping Stones, and Cryptojacking. Scenarios such as stepping stone behavior or Cryptojacking previously had no available datasets for study despite need from academic and industrial researchers.

We provide a complete list of implemented scenarios in Table 7.1.

7.5 Validation experiments

A framework that generates network traffic does not necessarily provide realistic and useful data. To evaluate the utility of our Docker framework, we construct a series of experiments. We have two goals in mind. First, we want to demonstrate that the traffic generated is sufficiently representative of real-world traffic. Second, we want to demonstrate that having a framework to continually generate data compared to static datasets benefits evaluating the efficacy of intrusion detection systems.

The first experiment provides a general verification of the reproducability of our framework, which is required for guarantee the ground truth of the produced data. The second experiment demonstrates that the WAN-characteristics we emulate for our data make it quasi non-distinguishable from real WAN traffic. Our third experiment then demonstrates the advantage of unlimited data generation capabilities for training ML-based traffic classification.

7.5.1 Reproducible scenarios

To provide ground truth, we have to guarantee that our implemented scenarios and subscenarios are consistent and reproducible upon repeated execution. This applies both to consistency for external influences on the host, such as increased computational load, as well as internal consistency of the implemented script execution.

Name	Description	#Ssc.
Ping	Client pinging DNS server	1
Nginx	Client accessing Nginx server	2
Apache	Client accessing Apache server	2
SSH	Client communicating with SSHD server	5
VSFTPD	Client communicating with VSFTPD server	12
Scrapy	Client scraping website	1
Wordpress	Client accessing Wordpress site	1
Syncthing	Clients synchronize files via Syncthing	1
mailx	Mailx instance sending emails over SMTP	2
IRC	Clients communicate via IRCd	2
BitTorrent	Download and seed torrents	3
SQL	Apache with MySQL	2
NTP	NTP client	2
Mopidy	Music Streaming	5
RTMP	Video Streaming Server	1
WAN Wget	Download websites	5
SSH B.force	Bruteforcing a password over SSH	3
URL Fuzz	Bruteforcing URL	1
Basic B.force	Bruteforcing Basic Authentication	2
Goldeneye	DoS attack on Web Server	1
Slowhttptest	DoS attack on Web Server	4
Mirai	Mirai botnet DDoS	3
Heartbleed	Heartbleed exploit	1
Ares	Backdoored Server	3
Cryptojacking	Cryptomining malware	1
XXE	External XML Entity	3
SQLi	SQL injection attack	2
Stepstone	Relayed traffic using SSH-tunnels	2

Table 7.1: Currently implemented traffic scenarios along with the number of implemented subscenarios

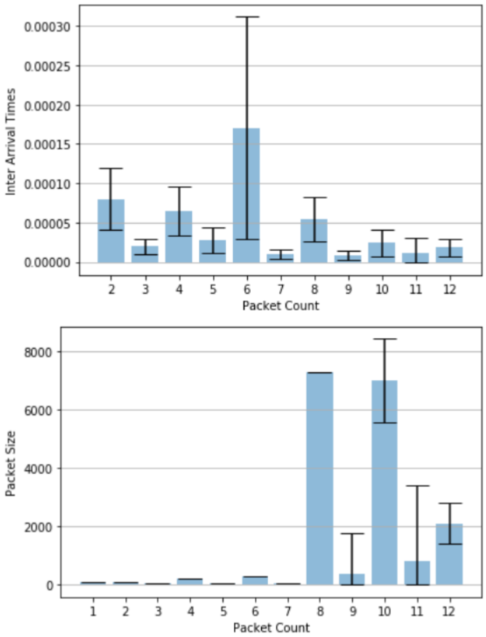


Figure 7.5: Means of IATs & packet sizes along with standard deviation bars for the first twelve packets in the Apache scenario.

It is impossible to guarantee that each scenario will produce a truly 'deterministic', or repeatable, output due to differences in network conditions, computational times, or input. Instead, we aim for our data to be *reproducible up to networking and computational differences.* This means that when running a scenario multiple times, we expect the quantities of most packets to be largely identical. We do expect some packets to exhibit greater variation due to non-determinism in the underlying protocols, Fig. 7.5 outlines this behavior in terms of interarrival times and packet sizes.

To measure how consistent our scenarios are, we generate 500 `.pcap` files for three different implemented scenarios, namely the Apache, the VSFTPD, and the SSH scenario. These were generated consecutively under different host CPU load. We did not apply any delays or other NetEm traffic controls.

We assess the consistency of a scenario across different `.pcap` files by comparing all generated `.pcap` files pairwize. We measure this by the similarity of the connections captured.

To test the similarity of two connections, we extract the sample distributions of the packet interarrival times and packet sizes overall, upstream and downstream. We define two connections as similar if the two distributions for each of these quantities pass an equality test. We use the two-sample Kolmogorov-Smirnov (K-S) test, a non-parametric statistical test for the equality of two continuous one-dimensional distributions [?], with a p-value of 0.01.

As all tested files passed this similarity test, we conclude that these scenarios yield consistent and reproducible results. As other scenarios follow the same setup and launch commands, we expect the results to stay the same as long as the involved containers are consistent in their behavior.

7.5.2 Exploring Artificial Delays

Most traffic our framework generates is transported over Docker's virtual network and therefore does not succumb to problems associated with normal network congestion, such as packet loss, corruption and packets arriving out of order. A realistic dataset should include these phenomena, which is why we developed wrapping scripts that allow us to artificially add delays as well as packet loss and corruption, using NetEm. Choosing the parameters is not straightforward; it is not clear how close to real-world traffic such network emulation techniques are. This is especially true for packet delays, which are described by continuous distributions and often have temporal correlation.

Furthermore, the high effective bandwidth of the Docker virtual network resulted in traffic with extremely short inter-arrival times (IATs, defined as the time between two packet arrivals). Therefore, we devote considerable time to demonstrating that it is possible for traffic generated by our Docker framework to conform to real-world IAT distributions when altered using NetEm.

Datasets

We create two classes of datasets, one which is representative of 'real-world' traffic, and one which has been generated from our Docker framework. For simplicity, we only consider datasets consisting of FTP traffic.

For the real-world dataset, we set up a containerized VSFTPD server running on a *Google Compute* virtual machine located in the Eastern United States, and a containerized FTP client on our local host. We then ran a series of our scripted interactions between the two machines, generating 834 megabytes of data in 250964 packets. These interactions consisted of several FTP commands with various network footprints. We collect all data transmitted on both the server and the client. We call this data the *Non-Local* dataset.

We then repeat this process using the same container setup, but across the Docker virtual network on a local machine. We repeat this process several times, generating several *Local* datasets under a variety of emulated network conditions, discussed in Section 7.5.2. Our *Local* datasets vary slightly in size, but are all roughly 800 megabytes with 245000 packets.

Methodology

NetEm allows us to introduce packet delays according to a variety of distributions, namely uniform, normal, Pareto and Paretonormal[1]. Furthermore, NetEm adds delays according to modifiable distribution tables, and so it is trivial for us to add a Weibull distribution, which along with Pareto distributions have been shown to closely model packet IATs [**?**, **?**]. In total, we test the efficacy of four distributions to model inter-arrival times — normal, Pareto, Paretonormal and Weibull.

We generate several *Local* datasets by delaying traffic according these distributions, performing an exhaustive grid search over their means and standard deviations. Initial experiments revealed that introducing delays with a mean in

[1] This Paretonormal distribution is defined by the random variable $Z = 0.25 * X + 0.75 * Y$, where X is a random variable drawn from a normal distribution and Y is a random variable drawn from a Pareto distribution.

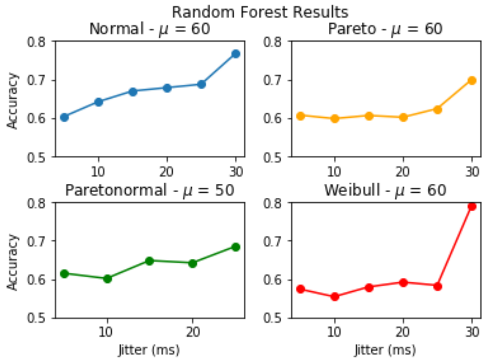


Figure 7.6: Results of Random Forest Classifier for a given distribution at the best performing delay mean μ. Note that a score of .5 indicates total indistinguishability.

the range of 40 ms to 70 ms produced the best results. Setting the jitter of the distribution too high resulted in the repeated arrival of packets out of order, therefore we further limit the grid search to jitter values in 5ms intervals up to half of the value of the mean. In total, we generate 88 *Local* datasets.

Our goal is to discover the *Local* dataset whose packet timings most closely resemble those of our *Non-Local* dataset. To do this, we extract the IATs and packet sizes from our datasets on a packet-by-packet basis and store these results in arrays. We measure the similarity between two of these arrays by training a Random Forest classifier to distinguish between them. We say that if the Random Forest correctly classifies each packet with a success rate of only 50% then it is no better than randomly guessing and, as such, the inter-arrival times of these two arrays are indistinguishable from one another for the Random Forest.

To perform this measurement, we concatenate one *Local* dataset array with our *Non-Local* dataset array, label the entries and then shuffle the rows. We proportion this data into a training set and a testing set using an 80-20 split. We then feed this training data into a Random Forest with 1000 trees and fixed seed, and then record the accuracy of this Random Forest on the test set. We repeat this process for every single *Local* dataset.

Results

Table 7.2 summarizes the values of the mean and jitter for a given distribution that produced the worst results from the random forest classifier.

Distribution	Mean	Jitter	RF Accuracy
No Delays (Baseline)	0	0ms	0.8176
Constant Delay	40ms	0ms	0.6730
Normal	60ms	5ms	0.6028
Pareto	60ms	10ms	0.5979
Paretonormal	50ms	10ms	0.6015
Weibull	60ms	10ms	0.5540

Table 7.2: Worst Random Forest accuracy rates for a given distribution

To establish a baseline, we compare the traffic generated from our Docker scenario to that of the *Google Compute* data with no added delays. In this case, the Random Forest was able to distinguish between the two datasets, achieving an accuracy of over 90%. The classification accuracy is worsened considerably by introducing network delays, with the best results being achieved using a Weibull distribution with a mean of 60 ms and a jitter of 10 ms, leading to an accuracy of just 55%. Results for Pareto and Weibull distributions seem to yield consistent results for differing jitter values. Although not completely indistinguishable, this proves that using NetEm we can emulate WAN properties very closely.

7.5.3 Advantages of Dynamic Dataset Generation

Having examined whether our Docker framework is capable of emulating real-world IATs, we explore their utility in traffic classification to demonstrate the advantages that our framework provides compared to static, unlabeled datasets.

Machine-learning techniques are a popular tool for traffic classification, with many successful published classifiers. Furthermore, inter-packet arrival times have been shown to be a discriminative feature [**?**, **?**]. However, these methods considered datasets consisting of completed traffic flows, limiting their use in, say, a stateful packet inspector. On-the-fly classifiers are also successful. Jaber et al. [**?**] showed that a K-means classifier can classify flows in real-time solely based on IATs with precision exceeding 90% for most protocols within 18 packets. Similarly, Bernaille et al. [**?**] demonstrated that a K-means classifier can precisely classify traffic within five packets using only packet size as a feature.

However, Jaber et al. [**?**] only evaluated their traffic classifier with training and testing data drawn from the same dataset containing traces of a single network; there is no measure of how this model may generalize to other networks with differing conditions. Furthermore, they were limited to using unsupervised machine learning algorithms to classify their traffic as their datasets had no ground truth.

We attempt to replicate these results within our Docker framework with some adjustments. As we can generate a fully accurate ground truth, we attempt to segregate application flows based on their packet IATs using supervised learning techniques. Moreover, we then measure this model's ability to generalize by expanding our dataset to include traffic from networks with differing bandwidth and latency.

Data & Preprocessing

Our goal is to measure a classifier's ability to generalize across datasets. Therefore we construct two datasets using our Docker framework, both containing the same number of network traces from the same containers.

For our first dataset, we generate `.pcap`-files, each containing traffic from one of 16 different classes: HTTP (Client & Server), HTTPS (Client & Server), RTMP (Client, Server & Viewer), SSH (Client & Server), FTP (Client & Server), IRC (Client & Server), SMTP, SQLi and DoS traffic. To prevent class imbalance, we generate 200 `.pcap`-files for each of the 16 classes, resulting in 3200 total files. To more accurately emulate potential network conditions, we use our NetEm scripts to apply a unique delay to every container involved in a scenario. These delays follow a Pareto distribution with random mean between 0 and 100 milliseconds and random jitter between 0 and 10 milliseconds. We then preprocess this data by removing all but the first 12 packets of each `.pcap`-file. We extract the 11 inter-arrival times separating the 12 packets, which act as our feature vectors. We collect these feature vectors for each class along with a class label, and store collected feature vectors from all 3200 `.pcap`-files in a 12 x 3200 array. We call this our *Primary* dataset.

We then repeat this process to generate a second dataset, changing the properties of our emulated network. Again, we delay all traffic using a Pareto distribution, however, this time we select a random mean in the range of 100 to 500 milliseconds and random jitter between 0 and 50 milliseconds. The subsequent preprocessing of our data remains unchanged. We call this our *Secondary* dataset.

Methodology

First, we attempt to reproduce the results presented by Jaber et al. [**?**] by training a Random Forest with 100 trees to classify application flows based on packet IATs. We do this by proportioning our Primary dataset into training and testing sets using an 80-20 split. We then train and test our Random Forest repeatedly, first considering the classification accuracy based on the IATs of only the first two packets, then the first three packets and so on, up to 12 packets. We record the resulting confusion matrix for each round and calculate the precision and recall rates of our classifier.

Having trained the classifier, we measure its ability to generalize by repeating the above experiment, but replacing the test set with the Secondary dataset.

Results

After each run of our Random Forest on our Primary dataset, we gather the True Positive (T_P), False Positive (F_P) and False Negative (F_N) rate for each class. We then calculate their precision, defined as $\frac{T_P}{T_P+F_P}$, and recall, defined as $\frac{T_P}{T_P+F_N}$, values. in Fig. 7.7, we see that our average precision and recall across the classes exceeds 0.9 after 10 IATs. Furthermore, after 12 packets our DoS and SQLi data is classified with precision and recall rates of 1.0 and 1.0 and 0.9462 and 0.9322 respectively.

These results does not hold when we test the classifyer on our Secondary dataset. As seen in Fig. 7.7, we see a substantial decrease in our average precision and recall rates, achieving a maximum of 0.5923 and 0.5676 respectively. Moreover, after four packets, increasing the number of IATs in our dataset provides little additional benefit. Although some services generalized well, such as IRC-client and IRC-server, others failed to be classified, with every single SMTP feature being classified as HTTP-client. We also see a substantial drop-off in the classification of malicious traffic, with the precision rates of DoS and SQLi data not exceeding 0.6.

These diverging results demonstrate the necessity of dynamic dataset generation for evaluation purposes. Researchers evaluating their methods only on a dataset with fixed properties such as the Primary dataset might receive overoptimistic results. The capability of generating two or more datasets with the same traffic classes, but otherwise differing properties, provides a more realistic evaluation.

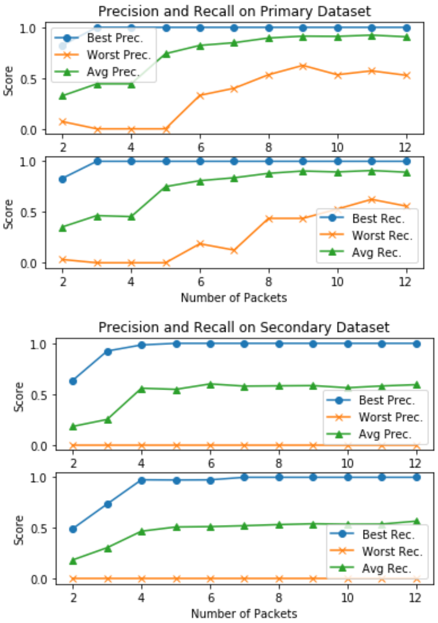


Figure 7.7: Results of Random Forest Classification on Primary dataset (Above) and Secondary dataset (Below)

7.6 Conclusions

In this paper, we outlined four requirements a modern dataset has to fulfil to strengthen the training of intrusion detection systems. We then proposed a Docker framework capable of generating network intrusion datasets that satisfy these conditions. The major design advantage of this framework are the isolation of traffic scenarios into separate container arrangements, which allows the extension of new scenarios and detailed implementation of subscenarios as well as the capture of ground truth of the computational origins of individual traffic events. Furthermore, containerization enables the generation of traffic data at scale due to containers being light-weight and easily clonable.

We verified the realism of the generated traffic and the corresponding ground truth information with two experiments, and demonstrated the usefulness of the framework in another experiment. Presently, our framework consists of 29 scenarios capable of producing benign and malicious network traffic. Several of these scenarios, such as the *BitTorrent* or the *Stepping-Stone* scenario, provide novel traffic data of protocols or behaviours that has not been widely available to researchers previously.

7.6.1 Difficulties and limitations

Our framework is building network traffic datasets from a small-scale level up by coalescing traffic from different fine-grained scenarios together. While this provides great insight into small-scale traffic structures, our framework will not replicate realistic network-wide temporal structures, such as port usage distributions or long-term temporal activity. These quantities would have to be statistically estimated from other real-world traffic beforehand to allow our framework to emulate such behavior reliably. Other datasets such as UGR-16 use this approach to fuse real-world and synthetic traffic and are currently better suited to build models of large-scale traffic structures.

Working with Docker containers can sometimes complicate the implementation of individual scenarios compared to working with VMs. Although several applications are officially maintained Docker containers that are free from major errors, many do not. For instance, in the *BitTorrent* scenario, most common command line tools, such as `mktorrent`, `ctorrent` and `buildtorrent`, failed to actually produce functioning torrent files from within a container due to Docker's union filesystem. Furthermore, due to the unique way in which we are using these software packages, unusual configuration settings are sometimes needed.

Lastly, capturing `.pcap`-files from each container can quickly exceed available disc space when generating traffic at scale. Depending on specific research requirements, it is advizable to add filtering or feature extraction commands to the scenario execution scripts to enable traffic preprocessing in real-time.

7.6.2 Future work

Our traffic generation framework is designed to be expandable and there are many avenues for future work. The continual development of scenarios and subscenarios would improve the potential realism of datasets generated using the framework. The addition of more malicious scenarios would enable a more detailed model evaluation and improve detection rate estimation. Another future improvement for framework is to add scripts that emulate the usage activity of individual scenarios by a user or a network.

Although ground truth for particular traffic traces is provided by capturing `.pcap`-files for each container individually, we have not implemented a labelling mechanism yet for the dataset coalescence process. Though not technically difficult, some thought will have to be put how such labels would look like to satisfy different research demands. Furthermore, the Docker platform provides the functionality to collect system logs via the `syslog logging driver`. We plan on implementing their collection in the future, where they could act either as traffic labels providing more ground truth details, or act as a separate data source that complements the collected traffic.

We wish to publish this framework to a wider audience, allowing for further modification. This will be done using a GitHub repository, which contains both the implemented capture scenarios as well as the corresponding container images.