FAKULTÄT FÜR ELEKTROTECHNIK UND INFORMATIONSTECHNIK
Faculty of Electrical Engineering and Information Technology

LSTM Autoencoders for Botnet Detection

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur/in

im Rahmen des Masterstudiums

Embedded Systems

eingereicht von

Jan de Bettignies

Matrikelnummer 00927468

an der Fakultät für Elektrotechnik und Informationstechnik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dr.-Ing. Tanja Zseby

Mitwirkung: Dipl.-Ing. Alexander Hartl

Wien, 09.12.2021

Jan de Bettignies

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit gemäß dem Code of Conduct – Regeln zur Sicherung guter wissenschaftlicher Praxis, insbesondere ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel, angefertigt wurde. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Wien, 09.12.2021

Jan de Bettignies

Abstract

Securing the Internet against malicious attacks is an ever evolving task. In this thesis, we aim at the detection of botnet traffic using two different machine learning methods: a random forest and LSTM autoencoder.

For our experiments, we use the public ISOT data set created by the Information Security and Object Technology (ISOT) research lab of the University of Victoria in Canada. It contains benign and botnet traffic and has been used in related work.

We select a random forest (RF) algorithm for its simplicity, ease of use, good classification performance and explainability. We compare it to a long short term memory (LSTM) autoencoder which is a variant of a neural network autoencoder. It is a more recent and complex algorithm that aims to extract not readily apparent information from the examined data, however the explainability is difficult.

We choose to compare a supervised approach (RF) with an unsupervised LSTM autoencoder and apply them to our data set using features as similar as possible in both machine learning models, with regards to the different characteristics of the two. The features can't be exactly the same because of the fundamental difference of a flow-based (random forest) vs. a packet-based (LSTM autoencoder) algorithm. A flow-based algorithm aggregates all packets in a set of statistical parameters which implies that a flow has to be terminated before calculating the characteristics. In contrast, a packet-based algorithm considers each individual packet and can, if necessary, try to compute a connection between those packets, even when a flow is still active. In the field of botnet detection, this can be of advantage as even an ongoing attack can be detected. Both algorithms are subject to a hyperparameter tuning. We also analyze which features influence detection for both the random forest and LSTM autoencoder algorithms.

The RF easily outclasses the LSTM autoencoder, achieving a ROC-AUC of 0.99 compared to 0.64 for the LSTM autoencoder. This shows that the LSTM autoencoder may not be the best choice for this task.

Kurzfassung

Der Schutz des Internets vor bösartigen Angriffen ist eine nicht enden wollende Herausforderung, die sich stetig weiterentwickelt. In dieser Arbeit geht es um die Erkennung von Botnet-Daten mit Hilfe zweier verschiedener Machine-Learning Methoden: einem Random Forest und einem LSTM Autoencoder. Für unsere Experimente verwenden wir den öffentlichen ISOT-Datensatz, der vom Forschungslabor Information Security and Object Technology (ISOT) der Universität von Victoria, Kanada erstellt wurde. Der Datensatz enthält sowohl "gutartigen" Datenverkehr als auch "bösartigen" Botnet-Datenverkehr, und wurde bereits in ähnlichen Arbeiten verwendet.

Wir wählen einen random forest (RF) Algorithmus, der sich durch seinen einfachen Aufbau, seine Anwenderfreundlichkeit, gute Klassifizierungsleistung sowie eine gute Erklärbarkeit der Ergebnisse auszeichnet. Wir vergleichen ihn mit einem Long Short Term Memory (LSTM) Autoencoder, einer Variante eines Neuronalen Netz Autoencoders. Diese Art von Algorithmus ist neuartiger und komplexer, und ist in der Lage, auch nicht augenscheinliche Informationen aus den untersuchten Daten zu extrahieren; allerdings ist die Erklärbarkeit der Ergebnisse sehr kompliziert. Wir vergleichen hierbei einen "überwachten" (supervised) Ansatz (RF) mit einem "nicht-überwachten" (unsupervised) LSTM Autoencoder und wenden beide Algorithmen mit möglichst ähnlichen Features für unseren Datensatz an. Da die beiden Algorithmen sich grundlegend unterscheiden, ist es nicht möglich, exakt dieselben Features für beide einzusetzen: bei einem Random Forest handelt es nämlich sich um einen flow-basierten Algorithmus, wohingegen eine LSTM autoencoder paket-basiert operiert. Ein flow-basierter Algorithmus aggregiert alle Pakete in ein Set und extrahiert statistischer Parameter, was implizit bedeutet, dass die Übertragung des gesamten Flows beendet sein muss, bevor seine Charakteristika berechnet werden können. Im Gegensatz dazu untersucht ein paket-basierter Algorithmus jedes einzelne Paket und kann, wenn nötig, auch bereits während der Übertragung eines Flows eine Verbindung zwischen den Paketen berechnen. In Bezug auf Botnet-Erkennung kann dies von großem Vorteil sein, weil es die Erkennung einer Attacke bereits im Moment ihrer Durchführung ermöglicht. Beide Algorithmen werden einem Hyperparameter-Tuning unterzogen. Außerdem untersuchen

wir, welche Features die Detektionsfähigkeit sowohl des Random Forests als auch des LSTM Autoencoders beeinflussen.

Der RF erreicht einen besten ROC-AUC Wert von 0.99 und ist hiermit dem LSTM Autoencoder (maximale ROC-AUC 0.64) in unserer Anwendung stark überlegen.

Acknowledgments

First and foremost, I am grateful to my thesis supervisor, Professor Tanja Zseby, and my tutor, Alexander Hartl. Their invaluable feedback and advice was always appreciated and their extraordinarily fast answers to my question at all times of the day or night were crucial in this endeavor.

I would like to extend my sincere thanks to Johanna for her unwavering support and belief in me, for proofreading and feedback. Getting through this thesis would not have been possible without her help.

Furthermore, I would like to thank all my friends and colleagues for their invaluable help on this journey: Birgit, Anna, Leopold, Stefan, Michael, Isabella, my colleagues of the "ZID-Raum Gang", and Sophie, Kathi, Pia, Alex, Emil, Clara and Ralph.

Finally, I would like to express my sincere gratitude to my family for their continued encouragement and support throughout my studies.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Over the last decades, the danger of botnets has only increased. A bot is *"a program that is installed on a system in order to enable that system to automatically perform a task or set of tasks typically under the command and control of a remote administrator"* [1], and a botnet is *"a concerted network of bots capable of acting on instructions generated remotely"* [1]. Botnets rely on infecting a large number of hosts to perform their malicious activity; this sets them apart from a targeted attack where only a few very specific hosts are attacked (an expamle would be the Stuxnet virus [2]). As a general rule, the more hosts a botnet can infect, the more successful (and to some extent resilient) a botnet is because it can leverage much more bandwidth for its activities [3]. A botnet can cause significant damage to public and private entities over its lifetime, whether through distributed denial of service (DDoS) attacks, downloading of additional malware, sending spam mail or theft of credentials. As an example, the ZeuS botnet was specialized in stealing banking information and responsible for more than $100 million in damages [4].

With the advent of the Internet of Things (IoT), the dangerousness of botnets has only increased [5]. Especially botnets specialized on compromising IoT devices have proven to be able to leverage significant attack and disruptive power [6]. These botnets rely on compromising any internet-connected device – a smart watch or a wireless camera for example. Botnet detection mechanisms able to detect the infiltration or infection of a host are crucial to mitigate damages from the botnet or to reduce the amount of spam sent over the Internet.

Detection of malicious activity, in this case botnet activity, can be compared to a cat-and-mouse game where the cat represents the detection algorithm while the mouse is the botnet. Both have to constantly evolve to keep an edge over their counterpart. Any solution can be outmatched and deceived by the threat actor given enough time and resources, rendering it less

effective to detect new and emerging threats.

These constant new and emerging threats have shown a limit to the practice of using Indicators of Compromise (IoC) for detection. Indicators of Compromise are artifacts that can be used *"[...] to indicate a computer intrusion and detect cyber-attacks in an early stage"* [7] and are observed on a computer or computer network. The response has been to turn to algorithms designed to detect malicious activity that do not use specific Indicators of Compromise [3]. A myriad of different algorithms and implementation strategies can be applied to this problem and not all of them have so far been investigated.

The evolving threat landscape makes any rigid, monolithic solution virtually impossible and very quickly obsolete [3]. A single, comprehensive solution for any and all threats is impossible, rather multiple different strategies have to be combined. The approach of this thesis is to concentrate on one specific category of threats, in this case botnets, and to examine different approaches for its mitigation. The drastic increase in computational power opens new possibilities to tackle the challenges faced by detection algorithms against malicious actors.

There are different types of machine learning algorithms which can be used to classify data. We select two of them, namely the random forest and a long short-term memory (LSTM) autoencoder. Since our data set is very large, the random forest is a promising approach, as it can handle large numbers of features in a data set [8]. Moreover, this type of algorithm has a simple architecture and short processing times and is therefore also applicable in low resource environments. However, Althibiti et al. [9] stress the limited ability of traditional machine learning algorithms, such as RF, to detect dynamic and complex cyber attacks on applications in comparison to deep learning approaches. A long short-term memory neural network is a deep learning approach; it has a much higher computational complexity, yet it is capable of remembering values of arbitrary intervals and consequently to predict known and unknown intrusions [9]. We will compare the random forest and LSTM autoencoder in order to see how they perform in detecting botnet traffic.

1.2 Research Questions

Our goal is to detect botnet traffic in our chosen data set and compare two machine learning algorithms, namely random forest and LSTM autoencoder, in their ability to detect this malicious traffic. For this we formulate the following research questions:

1. What detection performance can be achieved with a simple random forest classifier based on aggregate flow features?

2. Can the detection performance be improved if we use an LSTM autoencoder classifier based on per-packet features?

3. How does the choice of hyperparameter settings influence the detection performance of the two algorithms?

In order to assess the detection performance, we use different standard performance metrics such as accuracy, precision, recall, F1-score and ROC-AUC, and compare the results to those found in literature.

1.3 Approach

In a first step, it is necessary to find and choose a suitable data set that would accommodate the following requirements:

- it accurately represents a network,
- has a large enough sample size of both benign and malicious traffic,
- has the benign and malicious data labeled,
- the complete data set must be available for public use and
- contain botnet data from different botnets.

We select the data set created by the Information Security and Object Technology (ISOT) research lab from the University of Victoria, Canada [10] because it corresponds to all of the criteria mentioned above. The data set includes malicious flows from three different botnets: ZeuS, Waledac and Storm. A more comprehensive background on the different botnets can be found in Chapter 2. For our use, the data set has to be rearranged and the features needed for our experiments have to be extracted from the original files. There is no better imitation than the original, hence the ISOT lab used real traffic data as benign data. However, there is an ethical constraint on using malicious data "in the wild", and propagating such malicious data is illegal, therefore the malicious data is artificially generated and then combined with the benign data by the ISOT lab.
We will compare two different machine learning methods: a random forest algorithm and a LSTM autoencoder. The implementation of the random forest is already provided and only some changes to accommodate for the new data set are made. The LSTM implementation on the other hand is written from scratch. We select the random forest algorithm for its simple

architecture and short processing times, and the LSTM autoencoder for its capability of extracting complicated structures even when they are not readily apparent.
For each of the implementations, a baseline experiment is carried out against which the success of any hyperparameter change is compared to. Additionally, the random forest implementation is also used as a reference point.

1.4 Contribution

We compare two methods to detect malicious activity, especially botnets. We show that a random forest is a solid method for botnet detection, whereas a LSTM autoencoder did not perform as well in our experiments.
Our contribution in this thesis is:

- We examine the capacity of the random forest algorithm to detect malicious attacks by evaluating the following key metrics: receiver operating characteristic area under curve, accuracy, f1-score, precision and recall. We also investigate the relevance of specific features for the classification by calculating the Gini importance of a feature.

- Additionally, we perform an evaluation of the influence of hyperparameters for the random forest classifier's performance. The random forest algorithm benefits, as a general rule for this data set, from a higher tree depth but only up to a certain point. For a tree depth beyond this point, the metrics do not improve and may even decrease while the computation times only increase. We observe that the maximum minimum number of samples per leaf (MMSL) does not have a great impact in the data set we used.

- We also examine the ability of a three-layer LSTM autoencoder algorithm to detect malicious attacks by evaluating the following key metrics: receiver operating characteristic area under curve, accuracy, f1-score, precision and recall. We evaluate the model by calculating the threshold and the resulting mean squared errors of all features.

- After hyperparameter tuning, for the chosen data set, we find that the influence of different LSTM autoencoder hyperparameters is very small; it could therefore be possible to use a "standard" set of parameters and not have to tune the algorithm. This cuts down drastically the time to deployment of the LSTM autoencoder.

- We tried to reduce the needed number of packets by only considering the first 10 packets in a flow of a LSTM autoencoder, but due to the bad performance of the algorithm this

goal was not achieved. The small number of packets exchanged could have given the opportunity to react in time to prevent the malicious attacker from compromising the network.

- Both machine learning models are trained and tested on the same data set and evaluated by similar metrics in order to be able to compare them to each other as good as possible.
- We try to use features as similar as possible in both machine learning models, with regards to the different characteristics of the two algorithms. The features can't be exactly the same because of the fundamental difference between a flow-based vs a packet-based algorithm. In short, a flow-based algorithm aggregates all packets in a set of statistical parameters whereas a packet-based algorithm considers each individual packet and can, if necessary, try to compute a connection between those packets.
- Some artifacts and peculiarities are found in the ISOT [10] data set, specifically a negative inter arrival time in few specific packets. In addition, we notice that the interarrival times for benign packets are, in this data set, often higher than those of the malicious samples.

1.5 Structure

In chapter 2, we will give a short introduction to the different types and variants of botnets and an overview of their capabilities as well as an introduction to two machine learning algorithms; namely random forest and LSTM neural network. The state of the Art and current research will be presented in chapter 3. In chapter 4, the data set, metrics and implementation will be introduced. In chapter 5, all experiments will be shown. Chapter 6 will explain the results of the experiments described in the previous chapter. A discussion of these results is found in chapter 7. At last, chapter 8 will reach a conclusion.

Chapter 2

Background

There are multiple different approaches to detect malicious activity in computer networks, therefore a short overview shall be presented:

A first approach is based on an existing knowledge of a certain behavior and useful signatures (e.g. specifically named files in certain locations), thus called signature-based. By searching for known identities of each specific intrusion event signature based intrusion detection systems are efficient at identifying known attack patterns, however they always need to be updated regularly and are always only as good as their recency [11]. A good example for this technique can be found in *Bhatia et al.* [12]. Unfortunately, signature based systems are wholly incapable of detecting new and unknown botnets for they can only detect structures they already know [13] which leads us to other approaches, such as DNS-based detection systems. These techniques rely on detecting particular patterns of DNS information generated by either the bot or the bot master trying to communicate with each other which differ from patterns of DNS information in normal traffic. However, botnets (e.g., fast flux) managed to adapt themselves to the situation so that DNS systems are no longer effective in detecting them. Different types of DNS-based detection systems are introduced by *Schonewille et al.* [14], *Dagon et al.* [15] and *Choi et al.* [16]. As the process of analyzing DNS traffic is very complex and furthermore not able to provide information on botnet propagation, this type of systems is not applied regularly anymore. Another way to identify botnet C&C traffic is to use deep packet inspection. In this approach the payload is inspected; for this purpose, the single packets are inspected with regards to the content they transport on their payload. As each single packet is observed thoroughly, this approach takes a lot of time which makes it less applicable. *Göbel et al.* [17] for example created a system based on passive traffic monitoring to detect suspicious IRC nicknames, servers and server ports. *Strayer at al.* [18] use a two-stage system based on inspecting the payload content. A more recent approach that is also used in practice is to search for behavioral anomalies in a network. Here, the focus lies on abnormal patterns in network traffic, and not on the content of

the information being transmitted. The main advantage of this method is its ability to detect unknown botnet threats. This can happen by either attack behavior or operational behavior analysis. In attack behavior analysis the characteristics of attacks are examined, which means that this is only applicable after a botnet is already propagated, established and the attack is started. As this analysis interferes with the normal communication it is possible for the bot master to be aware of being observed [19]. In contrast, operational behavior analysis focuses on the bots' characteristics, C&C servers and communications or bot master behavior. In order to do so the network traffic is collected for a specific period and analyzed in order to identify bots or their activities [19]. More details about his approach can be found in 2.2. The reader shall be directed to *Chandola et al.* [20] for a much more in-depth look on the origins of anomaly detection not only in the context of Internet security.

2.1 Botnet typography

A botnet is *"a concerted network of bots capable of acting on instructions generated remotely"* [1]. A bot is *"a program that is installed on a system in order to enable that system to automatically perform a task or set of tasks typically under the command and control of a remote administrator"* [1] and this remote administrator is usually a malicious actor has taken command of this computer, unnoticed by the "original" owner of the computer. These bots are under the total control of the malicious actor and can be used for a number of tasks – sending spam mail, infecting other devices, participating in a DDoS attack, etc. [3] In recent time (see the Mirai botnet [21], [6]), the restriction that a botnet infects only desktop computers (PCs) has eroded, and increasingly any web-connected device, from PCs to mobile phones to surveillance cameras, can be transformed into a bot. One strategy to create a botnet is to create a so-called worm. RFC4949 describes a worm as the following: "A computer program that can run independently, can propagate a complete working version of itself onto other hosts on a network, and may consume system resources destructively." [22]. The important distinction to a virus is that a worm can and will operate independently of any input to infect as many other targets as possible.

A virus on the other hand does not share the self-replicating abilities of a worm, as such it must be delivered by other means to a target. Oftentimes this is done with phishing or drive-by download, with targeted attacks being used for more critical/sensitive infrastructures. A drive-by download is an unwanted and unknown download of malware.

A botnet can be structured according to two different principles: either centralized, hierarchical or a decentralized, peer-to-peer setup; hybrid botnets are also known [3].

A centralized botnet follows a very clear top-down structure starting with one or more defined

servers which are directly controlled by the malicious actors. At each subsequent level, the bots receive the commands from a single defined node while no connection is made between nodes on the same level. On the other hand, a peer-to-peer structure follows no such clearly defined rules, and any node can communicate with any other node. The command and control (C&C, aka C2) servers are interspersed in this network, and it is, looking from the outside, never exactly clear which nodes are the C&C servers. Hybrid botnets mix both structures [3].

Fast-fluxing is a method first observed on a large-scale malware in 2006 by the Strom network to increase persistence [23]. In essence, the domain name is associated to a large number of IP addresses which are changed extremely rapidly – an IP address can be changed every 300 seconds. As a consequence, IP address blocking is no longer a viable mitigation strategy against malware using the fast-flux technique.

Distributed Denial of Service (DDoS) attacks are attacks in which a very large number of devices simultaneously and repeatedly try to connect to a single known host in order to overload the host with connection attempts and deny any service from the host [24].

The data set used in the research contains three malicious actors that were introduced to a host of benign data: The Storm, Waledac and ZeuS botnets, which are described below.

2.1.1 Storm botnet

With the first detection made in January 2007, the Storm botnet rapidly propagated with the use of a self-replicating worm. At one point, over 2.6 million devices [25] were infected with the worm and together accumulated a computing power well beyond the computing power of the supercomputers at the time. The botnet used a modified version of a peer-to-peer communication protocol to communicate between the different bots. Additionally, a fast-flux DNS changing method was also implemented to make tracking of the botnet more difficult. To render any investigation more difficult and potentially dangerous, a self-defense mechanism was implemented: an automatic DDoS attack was launched against any entity repeatedly querying addresses of the botnet. Beginning in early 2008, the botnet was used in phishing attacks and at the same time introduced encrypted communication to enable the botnet to resell parts of the network or its services [23]. Its primary role was the distribution of malware and pharmaceutical spam, a characteristic it shares with the Waledac botnet. The botnet's decline started in 2008, when German security researchers successfully reverse-engineered the botnet and published a tool to take over certain parts of the botnet.

2.1.2 Waledac botnet

Sharing similarities with the Storm network, the Waledac botnet first appeared in December 2008. It is based on a peer-to-peer communication structure controlled by over 270 Internet domains [26]. As with the Storm botnet, this worm automatically starts DDoS attacks against any entity investigating it. The botnet was a major spam distribution network while additionally segments of the botnet were rented out to interested parties for their own spam distribution (rates were 200−500 per million spam messages). Further similarities, for example major code reuse, were shared with the Kelhios botnet. Beginning in February 2008, Microsoft initiated the take-down of the botnet by seizing all internet domains used as C2 servers [27].

2.1.3 ZeuS botnet

The ZeuS (aka Zbot) malware family first appeared in July 2007 and is characterized as a Trojan malware. It was distributed via drive-by downloading or phishing attacks. After infection, a botnet comprised of compromised hosts in the target network is set up. The C2-server is controlled directly by the attacker. This botnet can be used to launch attacks against other targets or simply spread in the victim network to gain intelligence about the network and/or exfiltrate data. The gained intelligence from the initial attack vector can then be used for follow-on attacks including ransomware (CryptoLocker) or theft of banking details (ex.: keystroke logging). The take-down by the FBI (Federal Bureau of Investigation; a federal law enforcement agency of the United States of America) in 2010 prompted the release of the source code by the author. This source code was then used in multiple successor versions by different threat actors, the biggest being GameOver ZeuS by the original author [28].

2.2 Machine learning algorithms

Our aim is to increase security in the connected world. Therefore, we need to find a way to differentiate between benign and malicious traffic. In our case we define malicious traffic as any data generated or received by bots with a different intention than legitimate users.
There is a plethora of approaches to identify and categorize malicious traffic such as simple statistics, machine learning algorithms, rule based algorithms, indicators of compromise based algorithms, etc. [3] The advantage of machine learning algorithms is that they are able to learn from known data and subsequently categorize unknown data [9]. For our study, we choose two different machine learning algorithms, namely RF and LSTM autoencoder, in order to not only examine but also to compare them. The underlying principles and characteristics of these

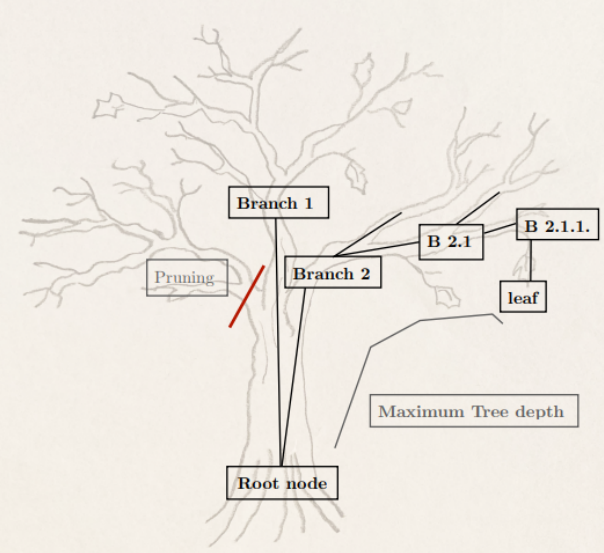

Fig. 2.1: *Decision tree with all important keywords and parameters.*

algorithms will be explained in the following paragraphs.

2.2.1 Random Forest

One way to make decisions is answering a series of simple closed questions arranged in the form of a so-called decision tree. Starting from a root node, each answer to a closed question leads to a next one up to a maximum number of questions, called the maximum tree depth.
Each of the branches has its ending point in a so-called leaf, where a decision is finally made; in our case the traffic being categorized as either benign or malicious. In order to ensure that each leaf is reachable, a minimum number of samples has to reach this leaf in order to be counted as a valid result. This minimum is called the minimum number of samples per leaf (min samples leaf, hereafter MSL). Depending on how wide or narrow a tree is desired to be, the MSL can be increased in order to eliminate the branches with only very few samples reaching the leaf. This process is called pruning [29]. What has just been described is illustrated in figure 2.1

One single decision tree can be prone to errors. Therefore, by using a great number of different, independently trained decision trees (like trees in a forest) and subsequently pooling their results, we can reduce the overall decision error. This multitude of decision trees is called a random forest. Each decision a single tree reaches is like a “vote”; all trees voting for a category leads to the overall decision “winner” who gains the most votes by number. This final voting decision is then the result of the random forest algorithm, as shown in figure 2.3 [30].

A random forest algorithm is a supervised machine learning algorithm. These types of algorithms are based on the principle that a data set with known results is fed to the algorithm to train it. Afterwards, the results of the algorithm can be compared to the already known

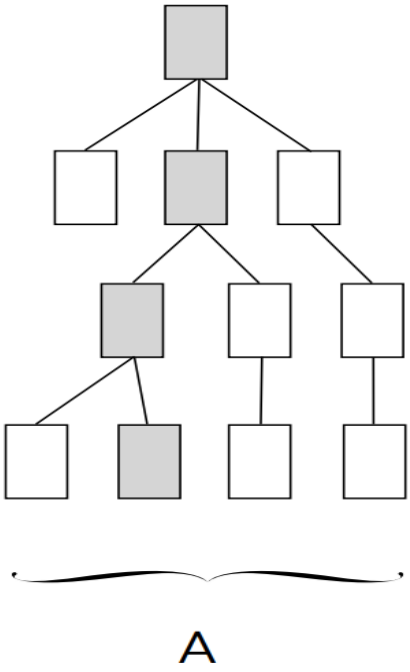

Fig. 2.2: *Decision tree.*

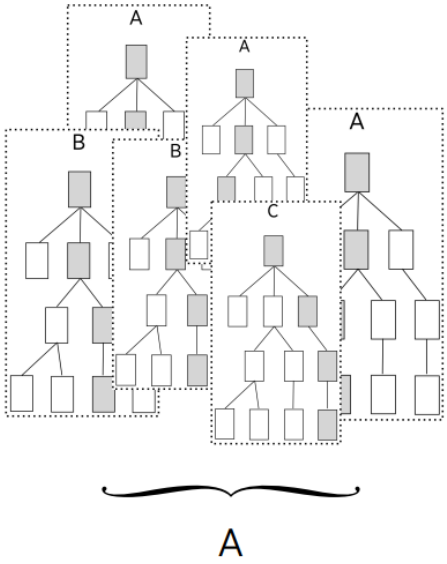

Fig. 2.3: *Random forest algorithm. Each decision tree casts one vote (A, B or C), the final classification is decided by simple majority. In this case with 3 votes for A, 2 for B and 1 for C the final decision is A.*

correct results, therefore this method is called supervised [3]. One characteristic of a supervised algorithm is that for proper training and testing labeled data is necessary. Therefore, it works only for previously known attacks or attack profiles. Random forests base their decisions on an entire flow, in our case all packets sent between two specific hosts and ports in a certain range of time from connection establishment to disconnection. In order to be applicable to a random forest, the data has to be aggregated, i.e. an entire flow is condensed into a certain number of metrics/ characteristics (e.g. number of packets sent, total number of tcp flags,...). The algorithm then decides on the basis of these characteristics, not on the single packets. Due to this necessary aggregation, the algorithm is not able to perform real time predictions for it can only start assessing after a flow has terminated.

The main advantages of using the method of a random forest are that the underlying principle is simple to comprehend and, especially in decision trees, the results can be retraced easily. Also, it is not complicated to calculate the importance of individual features in respect to the complete data set. However, the interpretability of a random forest is harder than of a single decision tree owing to the fact that a multitude of trees are taken together and pooled.

The computational complexity is rather low and usually accompanied by a short computation time. In particular, training times can be manageable in comparison to other machine learning algorithms, such as e.g. neural networks. Moreover, a random forest algorithm can be used both in a regression and in a classification role, making it a versatile machine learning method.

2.2.2 Long Short-Term Memory (LSTM) Neural network

A neural network aims to replicate the way a brain works: just like synapses send an electrical signal to trigger neurons, the algorithm triggers with a certain probability (called weight) a node, which in turn can trigger other nodes ([31]). *Abu-Nimeh et al.* [8] describe the concept very clearly: *"The neurons are not powerful by themselves, however, when connected to others they can perform complex computations. Weights on the interconnections are updated when the network is trained, hence significant interconnection play more role during the testing phase."* The distinguishing features between different feed-forward neural networks are the number of neurons in a layer and whether all neurons of one layer can trigger all neurons of the subsequent layer (called dense or fully-connected) or not (called sparse connection). A feed-forward neural network is also not able to process sequential data, which is the main drawback of this type of neural network.

To process sequential data, the neural network has to be expanded with a feedback loop that has the ability to commit to memory and recall previous states and implement them to the current input. When a feedback loop with these characteristics is implemented to a feed-forward neural network, the result is a so-called recurrent neural network. The most important part of a recurrent neural network is its hidden layer where the crucial part of the work is done. These hidden units can be regarded as a storage of the whole network where the end-to-end information is remembered. This approach can be used for supervised classification learning. As *Le et al.* state: *"The preceding output is also related to the current output of a sequence, and the nodes between the hidden layers are no longer connectionless; instead, they have connections. Not only the output of the input layer but also the output of the last hidden layer acts on the input of the hidden layer."* [32]

The recurrent neural network suffers from two problems as a consequence of the introduction of the feedback loop. One problem is the vanishing gradient descent, where events become smaller over time and eventually disappear. The other is the exploding gradients problem where the long-term components (i.e. weights of the feedback loop) grow exponentially fast. [33] To overcome the vanishing gradient descent problem, a feedback loop with a non-linear loss function is introduced to enable the recurrent neural network a special memory of previous events [34]. A neural network with all of these aspects combined is then called a Long Short-Term Memory (LSTM) neural network. There are no better words to explain and summarize LSTMs than *Bontemps [35]: " The LSTM node structure enables a phenomenon called backpropagation*

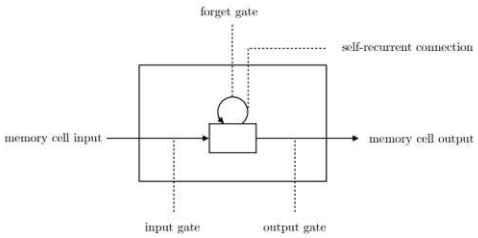

Fig. 2.4: *A single cell of an LSTM.*

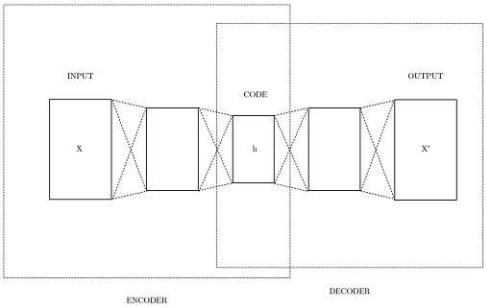

Fig. 2.5: *Schematic of an exemplar 2-layer autoencoder. The hidden layer (a.k.a. code) h is the innermost layer.*

through time. By calculating for each hidden layer the partial derivatives of the output, weight and input values, the system can move backwards to trace the evolving error between real output and predicted output. Afterwards, the network uses the derivative of this evolution to adapt its weights and decrease prediction error. This learning method is named Gradient Descent. [...] a prediction is made focusing on two features: the value of a sample and its position at a specific time. This means that two input samples at different times may have the same value, but their outputs will very probably differ. It is because a LSTM RNN is stateful, i.e. has a "memory", which changes in response to inputs."

Another type of an artificial neural network is a so-called autoencoder. In its simplest form it consists of an encoder mapping the input into the code and a decoder mapping the code to a reconstruction of the input connected by at least one hidden layer. Input and output layer have the same number of neurons (nodes) and it is the latter's purpose to reconstruct the inputs by minimizing the difference between the two layers, therefore it is trained to minimize the reconstruction errors. As it is a feedforward neural network, the training is through backpropagation of the error. For anomaly detection, autoencoders are trained to learn to replicate the training data's characteristics [36]. If confronted with anomalies, the reconstruction error increases, therefore this error is used a score for anomaly detection [37]. *Madani et al.* [38] describe *"Due to the bottleneck in the representation layer, the autoencoder is forced to encode only approximately the underlying concepts that resemble the input. As a result, the model often learns useful properties of the input data in order to achieve necessary reconstruction ability."* An example of an autoencoder is shown in Figure 2.5.

We can combine multiple layers of neural networks in a certain way to create an LSTM autoencoder. The decoder then takes this compressed information and tries to reconstruct the original data [39]. As such, a LSTM Autoencoder offers a new approach to the detection of botnets in a network by combining the advantages of a LSTM neural network with the advantages of an

Random Forest	LSTM autoencoder
supervised	unsupervised
flow-based	packet-based
aggregated data	raw data
after flow end	during flow
some explainability	low explainability
tree-based	artificial neural network-based
labeled data for training/testing	no labeled data needed

Tab. 2.1: *Differences and similarities of a random forest and a LSTM autoencoder*

autoencoder. These include: capacity to extract critical information, recollection of previous events.

In contrast to a random forest, an LSTM autoencoder is an unsupervised machine learning algorithm. According to *Xia et al.* [39], *" [a] classic autoencoder is a feedforward fully-connected neural network, where the number of neurons in the input layer and the output layer are the same and where there are much fewer neurons in the hidden layers than in the input and output layers."* In our case, we replace the mentioned feedforward neural network by a LSTM neural network. The algorithm works on a packet-based principle, analyzing all the packets individually within each flow. Because the LSTM autoencoder does not need to aggregate a flow, it can perform a real time prediction and classification and even allows a user to terminate malicious flows instantly from the outside during an ongoing attack. As any outliers are reconstructed with a higher error, anomalies including previously unknown attacks can be detected. However, it is not possible to follow the algorithm's reasoning behind a decision, it is a "black-box" where only the input and output are known. This means that the decision is hardly explainable.

Through the process of compressing and decompressing the decoded data is compared to the data before it had been encoded, however it is not compared to an external source and therefore not supervised. Because it does not need supervision, we do not need to label the data for training or testing which spares time and resources in the preprocessing stage. Also, the flows do not need to be aggregated. The main drawbacks of this method is the great computational complexity, a classifier decision that is difficult to explain and the longer computational time an experiment takes. Its great benefit is the capability of extracting not readily apparent information and memorizing a sequence of events.

A quick comparison of both algorithms can be seen in 2.1. Both these types of algorithms work very well for machine learning, so we decided to compare them with the same data set and see how each of them performs.

Chapter 3

State of the Art

At the moment, there is no consensus about a data set to use, methods to compare against or even metrics to compare the methods. Many different groups have used different data sets and different metrics in their research which makes it very difficult if not nigh impossible to define a standard to which one's results can be compared to. What makes the situation even more complex is the diverging nomenclature applied to the metrics.

The following authors examined the performance of random forest, sometimes comparing to other machine learning methods, in botnet detection:

In 2013, *Garg et al.* [40] implemented three different algorithms: one Naive Bayesian (NB), a k-nearest neighbor and a C4.5 decision tree. All of them were tested on the ISOT [10] data set and achieved a recall of 0.99, 0.995 and 0.90 for the NB, k-NN and C4.5, respectively.

In 2014, *Stevanovic and Pedersen* [41] implemented eight different algorithms: Naive Bayesian (NB), Bayesian Network classifier, logistic regression (LR), an artificial neural network (ANN), a support vector machine with linear kernel (SVMlin), the C4.5 decision tree (DT), random tree (RT) and random forest (RF). All experiments were done on the ISOT [10] data set. The RF showed the overall best results (precision, F1-score, Matthews correlation coefficient), only for the recall the NB showed better results, whereas the LR mostly performed least.

In the same year, *Beigi et al.* [29] decided to implement a C4.5 decision tree with reduced error pruning algorithm. For their experiments, they used a combination of the ISOT [10] and ISCX [42] data sets and achieved a recall of 0.75 with a false positive rate of 0.25.

Three years later, in 2017,*Alejandre et al.* [43] implemented a C4.5 decision tree algorithm and tested it on a combination of the ISOT [10] and ISCX [42] data sets. They reported a recall of 0.9946 and a false positive rate of 0.0057, picturing the good performance of the decision tree. In comparison to *Beigi et al.* [29], the results are far better which could be due to the reduced error pruning Beigi used back in 2014.

Four different algorithms were implemented by *Wai et al.* [44] in 2018: support vector machine (SVM), decision tree (DT), random forest (RF) and a 3-layer Multi-Layer Perceptron (MLP) with 432 neurons in the first layer and 600 in the other two. Tested with a combination of the LBNL [45], ISCX [42] and Stratosphere IPS [46] data sets, the best result in respect to recall and false positive rate (FPR) was achieved using the MLP.

Also, in 2018, *Abraham et al.* [11] implemented five different machine learning algorithms: a logistic regression (LR), a Naive Bayesian (NB), a support vector machine (SVM), a random forest (RF) and a feed-forward neural network (ff-NN). All algorithms were tested with the Stratosphere IPS [46] data set. The F1-scores as well as the AUC showed worst results in NB, whereas the highest scores were achieved by RF.

At the same time, *Bahsi et al.* [47] created a custom data set to test two different machine learning algorithms: a decision tree and a k-nearest neighbor. Unfortunately, they did not give any details regarding the exact nature of either of the algorithms. They report an accuracy of 0.9897 with the decision tree and 0.9497 with the k-NN algorithm.

In December 2018, *Haq and Singh* [48] implemented three machine learning algorithms: k-means clustering, C4.5 decision tree and a combination of both. All were tested on a combination of ISOT [10] and CTU13 [46] data sets. They used a peculiar metric consisting of the sum of false positives and false negatives divided by the total number of packets. They achieved a rate of 0.549094 with the k-means clustering, 0.097277 with the C4.5 and 0.219069 when both are combined. As this metric is said to picture the "percentage of incorrect instances" the C4.5 decision tree shows the best result in achieving the lowest score.

Over the last quinquennial, following authors examined the performance of LSTMs, autoencoders and LSTM autoencoders, sometimes comparing them to other machine learning methods, aiming to detect DDoS attacks, botnets, general intrusions as well as anomalies:

In 2016, *Bontemps et al.* [35] implemented a LSTM with a hidden layer of 23 neurons. Using four different thresholds, they analyzed the algorithm with the KDDCup99 [49] data set. At a threshold of 0.69, a recall of 0.86 with no false positives was achieved. Lowering the threshold to 0.66 gave a recall of 0.94 but two false positives. With a threshold of 0.62, the algorithm achieved a recall of 0.98 with 16 false positives. Lastly, a threshold of 0.52 resulted in a recall of 1 but 32 false positives.

In the same year, *Kim et al.* [50] implemented a LSTM with a hidden layer of 80 neurons. With the KDDCup99 [49] data set, the algorithm achieved an accuracy of 0.9693 with a recall of 0.9888 and a false alarm rate of 0.1004.

Also, *Torres et al.* [51] implemented a 128 neuron LSTM and performed all experiments on

the CTU13 [46] data set in 2016. They reported an Attack Detection Rate (also known as recall) of 0.809 with a False Positive Rate (FPR) of 0.030.

In 2018, *Fu et al.* [52] implemented a long short-term memory neural network. Unfortunately, no details as to the exact parameters are given. Tested on the NSL-KDD [49] data set, the LSTM achieved a recall of 0.9885, accuracy of 0.9752 and false alarm rate of 0.0875.

In the same year, *Yin et al.* [53] implemented a 4-layer LSTM with 122 neurons in the first, 80 neurons in the second, 20 in the third and 3 in the last layer. Using the ISCX [42] data set, their algorithm achieved a precision of 0.6986, accuracy of 0.6900 and F-score of 0.6851.

Also in 2018, *Mirza and Cosan* [54] implemented eight different machine learning algorithms: LSTM, GRU, bidirectional LSTM (Bi-LSTM), 1-layer neural network, LSTM with last pooling, LSTM with max pooling, LSTM with mean pooling and a deep autoencoder LSTM. The LSTM, Bi-LSTM, and NN used a threshold of 0.08; the GRU a threshold of 0.006; the mean pooling LSTM a threshold of 0.079 and all others 0.076. The highest F1-score was observed in the deep autoencoder LSTM and LSTM with max pooling, the best AUC score of 0.9519 was achieved by the LSTM.

Using the CSIC 2010 [55] data set, *Althubiti et al.* [9]in April 2018 tested a long short-term memory (LSTM) neural network. The algorithm achieved an accuracy of 0.9997, precision of 0.995 and recall of 0.995.

A couple of months later, *Althubiti et al.* [56] implemented a 3-layer LSTM neural network with 10 neurons in layer 1, 6 neurons in the hidden layer and 5 in the output layer. They tested the algorithm with the CIDDS [57] data set and report an accuracy of 0.8483, precision of 0.8514, recall of 0.8834 and false positive rate 0.1722.

In early 2019, *Roopak et al.* [58] implemented four different machine learning algorithms: Multi-Layer Perceptron (MLP) with three layers, a 1-dimensional convoluted neural network (1d-CNN) with two layers, a 1-layer LSTM with 128 neurons and a combined 1d-CNN with LSTM. Used on the CICIDS 2017 [59] data set, they achieved the best accuracy and recall for the combination of 1d-CNN with LSTM, whereas the best precision was achieved by the LSTM alone.

In November 2019, *Gwon et al.* [60] implemented two long short-term memory neural networks: one with a single, 100-neuron layer and the other with a 100-neuron and a 10-neuron layer. The first LSTM achieved an accuracy of 0.9942 and a F1-score of 0.9947, while the second reached an accuracy of 0.9972 with a F1-score of 0.9975.

Kasongo and Sun [34] in 2020 implemented a 3-layer LSTM with a total of 90 neurons while withholding the exact distribution between the layers. They used the NSL-KDD [49] data set

and achieved a validation accuracy of 0.9951 with an F1-score of 0.9943 and a test accuracy of 0.8699.

In 2021, *Min et al.* [61] implemented a novel memory Autoencoder (MemAE) and tested it on three different data sets: the NSL-KDD [49],UNSW-NB15 [62] and the CICIDS 2017 [59]. Their modified autoencoder solves an oversimplification problem inherent to autoencoders by introducing a memory module which aims to prevent the autoencoder from reconstructing anomalous samples well. In their experiments, they achieved a ROC-AUC score of 0.9681 for the NSL-KDD data set, a score of 0.9113 for UNSW-NB15 and 0.9101 for CICIDS2017.

As a general rule, a classification LSTM which has often been used in these publications has a higher accuracy than a LSTM autoencoder for anomaly detection.

To complete the picture, the following authors who in the last five years used different approaches than random forest or LSTMs or LSTM autoencoders to detect intrusions, anomalies or botnets are presented shortly:

In 2016, *Al-Jarrah et al.* [63] implemented a randomized data partitioned learning model (RDPLM) and tested them using the ISOT [10] data set. The reported accuracy is 0.999845, with a recall of 0.9942 and a false alarm rate of 0.0009.

In 2017, *Yin et al.* [53] implemented multiple variants of a recurrent neural network (RNN) and tested them on the NSL-KDD [49] data set. One RNN with 80 hidden nodes and a learning rate of 0,1 achieved a training accuracy of 0.9981 and a test accuracy of 0.6855. Another one with 80 hidden nodes and a learning rate of 0,5 had a training accuracy of 0.9953, testing accuracy of 0.6467. False positive rates were all around 0.002 and recall between 0.1150 and 0.8349.

In the same year, *Koli and Chavan* [64] implemented a randomized data partitioned learning model (RDPLM) and tested it on the ISOT [10] data set. Their results were an accuracy of 0.99984, F-measure of 0.995 and FAR of 0.00008.

In Summer 2018, *Chen et al.* [65] implemented two neural networks: one convolutional neural network with 3 layers of 20 neurons each, and the same setup but combined with a decision tree. Using the CTU [46] and PeerRush [66] data sets, they achieved an accuracy of 0.947 and 0.986 respectively.

In December 2018, *Nomm and Bahsi* [67] created a custom data set with which to test their implementations of a SVM and Isolation Forest. The SVM achieved an accuracy of 0.9315 with a precision of 0.9627, while the isolation forest had an accuracy of 0.9561 and precision of 0.9860.

Based on the heterogeneous state of the art we decide to include many different metrics such that a comparison to at least some other works could be made. In Table 3.1, all the papers are summarized.

Paper	Year	Goal	RF/DT	LSTM	Others	data set
Stevanovic et al. [41]	2014	Botnet detection	x		x	ISOT
Wai et al. [44]	2018	Botnet detection	x		x	LBNL, ISCX, IPS
Abraham et al. [11]	2018	Botnet detection	x		x	IPS
Alejandre et al. [43]	2017	Botnet detection	x			ISOT, ISCX
Bahsi et al. [47]	2018	Botnet detection	x			custom
Beigi et al. [29]	2014	Botnet detection	x			ISOT, ISCX
Garg et al. [40]	2013	Botnet detection	x		x	ISOT
Haq et al. [48]	2018	Botnet detection	x		x	ISOT, CTU13
Roopak et al. [58]	2019	DDoS attack detection		x	x	NSL-KDD, UNSW-NB15, CICIDS17
Torres et al. [51]	2016	Botnet detection		x		CTU13
Yin et al. [68]	2018	Botnet detection		x		ISCX
Mirza et al. [54]	2018	Intrusion detection		x	x	not specified
Kim et al. [50]	2016	Intrusion detection		x		KDDCup99
Kasongo et al. [34]	2020	Intrusion detection		x		NSL-KDD
Althubiti et al. [9]	2018	Intrusion detection		x		CSIC2010
Althubiti et al. [56]	2018	Anomaly detection		x		CIDDS
Bontemps et al. [35]	2016	Anomaly detection		x		KDDCup99
Gwon et al. [60]	2019	Intrusion detection		x		not specified
Fu et al. [52]	2018	Intrusion detection		x		NSL-KDD
Min et al. [61]	2021	Anomaly detection		x	x	NSL-KDD, UNSW-NB15, CICIDS17
Yin et al. [53]	2017	Intrusion detection			x	NSL-KDD
Nomm et al. [67]	2018	Anomaly detection			x	custom
Koli et al. [64]	2017	Botnet detection			x	ISOT
Al-Jarrah et al. [63]	2016	Botnet detection			x	ISOT
Chen et al. [65]	2018	Botnet detection			x	CTU, PeerRush

Tab. 3.1: *Other works*

Chapter 4

Methodology

In this chapter, we first describe the methodology we use in this thesis. We give a short introduction on the data set and explain how we adjust it for both our machine learning algorithms. Afterwards, the metrics with which the outcomes are evaluated are described. Finally, we present how the implementation for both the random forest and LSTM autoencoder is accomplished.

4.1 data set

For the analysis, we use the **ISOT data set** [10]. The authors, the Internet research group of the University of Victoria, describe how multiple already existing publicly available data sets containing benign and malicious data were combined. The benign data was combined from the Traffic Lab at Ericsson Research in Hungary [69] and the Lawrence Berkeley National Laboratory (LBNL) [45]. Data from the LBNL consists of the network trace of short periods of up to one hour taken during five different days. The Traffic Lab data was captured during a 43-hour long measurement. The malicious data comes from a honeypot installed by the French chapter of the honeynet project [70]. These different data sets were then unified in a single data set to create the ISOT data set.

Before the data set can be used for our experiments, it has to be prepared. We will explain the steps taken for the random forest data set in section 4.1.1 and for the LSTM autoencoder in section 4.1.2. Since the original data set is available in a packet capture (pcap) format, we first have to extract all the features intended to be analyzed and convert them to a comma-separated value (csv) format. Since a pcap formatted file is packed-based, the conversion also has to aggregate the packets into their original flows to which the individual packets are then assigned. For this, we use go-flows [71], a tool provided by the Institute of Telecommunications of the TU Vienna.

4.1.1 Random Forest data set

Based on *Williams et al.* [72], we decide to select similar features as stated to have a reference to compare to. As the random forest algorithm is a flow-based method, we calculate the following 43 **features during the extraction** from the pcap files:

- source and destination transport port and protocol identifier
- Total count of packets sent (forward as well as backward)
- Total count of bytes sent (forward as well as backward)
- Minimum, mean, maximum, standard deviation and variance of the IP total length (forward as well as backward)
- Minimum, mean, maximum, standard deviation and variance of the inter arrival time measured in milliseconds (ms) (forward as well as backward)
- Total count of the different TCP flags, which are: SYN, ACK, FIN, PSH, RST, URG, CWR and ECE (forward as well as backward)

With the extraction of these features for both benign and malicious data, we reduce the 11GB pcap datafile to two csv files with 487MB in total, which correspond to flow-based features of 2,114,608 benign flows and 52,659 botnet flows. Both botnet and benign data are then labeled, shuffled together and split into two parts to create the training and testing data sets. The further process is explained in section 4.3.1. A list of all features used is found in table 4.2.

4.1.2 LSTM Autoencoder data set

For the LSTM autoencoder data set, the following parameters are applied for the **feature extraction**: bidirectional flows are enabled, idle timeout and active timeout are set to 300s. The 300s seconds are chosen because the default Idle Timeout of the TCP protocol is 300 seconds (or 5 minutes) [73], [74]. At the same time, the following features, inspired from [72], are saved per packets:

- destination port
- flow direction, assessed by the source and destination IP address
- inter-arrival time between one packet and the next packet
- total length of the IP packet

Name of feature	Minimum	Maximum
sourceTransportPort	0	65,535
destinationTransportPort	0	65,535
protocolIdentifier	1	17
packetTotalCount, fwd	1	1,346,271
octetTotalCount, fwd	28	1,978,876,600
min ipTotalLength, fwd	28	1,500
mean ipTotalLength, fwd	28	1,500
max ipTotalLength, fwd	28	1,500
stdev ipTotalLength, fwd	0	1,032.38
variance ipTotalLength, fwd	0	1,065,800
min interPacketTime_ms, fwd	-3,615,251	299,999
mean interPacketTime_ms, fwd	-3,516,144	299,999
max interPacketTime_ms, fwd	-3,516,144	299,999
stdev interPacketTime_ms, fwd	0	2,756,703.87
variance interPacketTime_ms, fwd	0	7,599,416,225,312
tcpSynTotalCount, fwd	0	100
tcpAckTotalCount, fwd	0	1,346,271
tcpFinTotalCount, fwd	0	19
tcpPshTotalCount, fwd	0	433,251
tcpRstTotalCount, fwd	0	100
tcpUrgTotalCount, fwd	0	1
tcpCwrTotalCount, fwd	0	6
tcpEceTotalCount, fwd	0	6
packetTotalCount, bwd	0	11,716
octetTotalCount, bwd	0	14,200,949
min ipTotalLength, bwd	0	1,237
mean ipTotalLength, bwd	0	1,237
max ipTotalLength, bwd	0	1,500
stdev ipTotalLength, bwd	0	780.65
variance ipTotalLength, bwd	0	609,408
min interPacketTime_ms, bwd	-21,597	255,554
mean interPacketTime_ms, bwd	0	255,554
max interPacketTime_ms, bwd	0	255,554
stdev interPacketTime_ms, bwd	0	112,632.92
variance interPacketTime_ms, bwd	0	12,686,174,184.5
tcpSynTotalCount, bwd	0	7
tcpAckTotalCount, bwd	0	11,716
tcpFinTotalCount, bwd	0	4
tcpPshTotalCount, bwd	0	2,708
tcpRstTotalCount, bwd	0	1
tcpUrgTotalCount, bwd	0	0
tcpCwrTotalCount, bwd	0	0
tcpEceTotalCount, bwd	0	0

Tab. 4.1: *Features used for the RF experiments and their minima and maxima before normalization. The negative values are discussed in section 4.1.3.*

Name of feature	Minimum	Maximum
destinationTransportPort	0	65535
flowDirection	0	1
interPacketTimeMilliseconds	-3604211.5	299999.99
ipTotalLength	28	1500
SYN	0	1
FIN	0	1
RST	0	1
PSH	0	1
ACK	0	1
URG	0	0
ECE	0	1
CWR	0	1
NS	0	0

Tab. 4.2: *Features used for the LSTM autoencoder and their minima and maxima before normalization. The negative values are discussed in section 4.1.3.*

- TCP flags (SYN, FIN, RST, PSH, ACK, URG, ECE, CWR, NS)

This reduces the original 11GB pcap file to a 2.5GB csv file consisting of 455,577 flows with a combined total of more than 149 million packets. Unfortunately, the packets are not distributed evenly within all of the flows; much to the contrary: a flow can have anything between one and 2,582,839 packets. Padding all of the flows to the maximum length of 2,582,839 packets would consume an unworkable amount of memory, hence the decision to pad all flows to a maximum of ten packets. At the same time, for flows with more than ten exchanged packets, we only use the first ten and discard all others. Table 4.3 shows an overview of the problem faced and the approximate distribution within the different subsets.

The botnet data is **split** from the complete data set and held in reserve for the testing set. Meanwhile, the rest of the data set (all benign data) is split into three different sets: a training set, a validation set and a testing set. The training set is given to the algorithm so it can learn to recognize characteristics of the benign data set. During this phase, the LSTM autoencoder sets the probability of being activated, the so-called weight (as shown in figure 2.4), for each neuron of each LSTM layer. This enables the autoencoder to remember similar benign data and to differentiate anomalies from it. Afterwards, validation is carried out using only benign data as well to check if the algorithm managed to learn the characteristics well. However, the algorithm is not supposed to learn the entire data set perfectly "by heart", called overfitting – it is only supposed to detect major characteristics. Neither overfitting nor underfitting (when the autoencoder is not capable of reconstructing the training data) are wanted.

To ensure that the algorithm doesn't over- or underfit, we use the validation data set verification of the fact. The validation data set is only used to get an idea of the performance of the model,

	Total number	Minimum	Maximum
Total			
Flows	455,577		
Packets	149,000,000	1	2,582,839
Training			
Flows	258,975		
Packets	2,589,750	10	10
Validation			
Flows	114,968		
Packets	1,149,680	10	10
Testing			
Flows	81,634		
Packets	816,340	10	10

Tab. 4.3: *Number of flows and packet distribution within flows*

ML model	Training	Validation	Testing
RF	benign + malicious	-	benign + malicious
LSTM autoencoder	benign	benign	benign + malicious

Tab. 4.4: *Overview of the data set parts used in the two models*

and is not used to calculate the threshold, which described in more detail in 4.3.2 Finally, the algorithm is challenged by the testing data set, which contains not only benign but also malicious (botnet) data. The aim of this stage is to verify the algorithm's performance in differentiating between these types of data [75].

The training set consists of more than 258,000 flows, with the validation set having more than 114,000 flows. Of the around 81,000 flows in the testing set, between 70 and 80 percent of those are botnet flows. The testing set is chosen randomly from the total number of flows available in the data set that remains (complete set \ training set \ validation set).

In order to test the questions made in Section 1.2, a LSTM autoencoder is created. In our case, three Long Short-Term Memory (LSTM) neural networks are chained together to create an encoder stage, to which is added a decoder stage with the same structure but in inverse order. This forms the 3-layer symmetric LSTM autoencoder that is subsequently used in the experiments.

Building upon a neural network, a key piece of information for a LSTM neural network is the change the input has undergone in time. As such, the input tensor of a LSTM autoencoder is three-dimensional: the flows x packets x features. Owing to the decision to limit the maximum number of packets per flow to 10, we can say that the LSTM has 10 timesteps to consider.

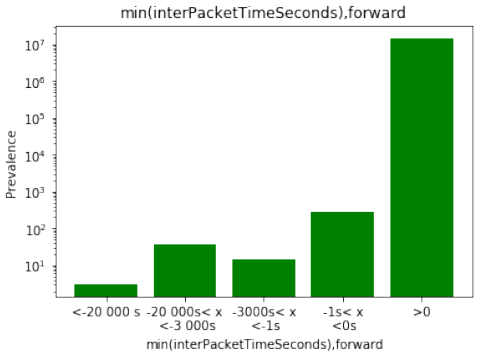

Fig. 4.1: *Overview of inter packet times in the raw ISOT data set.*

4.1.3 Artifacts in the data sets

We also observe that a few packets in the ISOT data set have a negative packet inter-arrival time which is in theory impossible because the inter-arrival time defines the time between subsequent packets which cannot be negative. However, within this data set we find a few hundred packets with negative inter-arrival times. We find a few packets with a negative inter packet time at around - 3,600 seconds that may be caused by an internal computer clock difference, such as a change in time zone or in daylight saving time. Negative inter-arrival times just below 0 (between - 0.01 and 0 sec) could be attributed to internal clock errors or synchronization problems. A negative inter packet time between the two former mentioned categories could also be attributed to wrong settings of computer clocks. However, we also see three packets with an inter packet time smaller than - 20,000 sec. This can happen when the system time and date are set completely wrong as it can happen occasionally in a real environment. All in all, only 300 of more than 15,000,000 packets showed this behavior. Since we discover this only after several experiments, we do not remove those values and continue working with the ISOT data set. The plot 4.1 gives the exact breakdown in ranges of time and the exact number at each range.

For the flow-based extracted data set, the two plots 4.2 and 4.3 show the distribution of the minimum inter-arrival time in seconds of the benign and malicious flows. All extracted flows (in csv format) will, compared to the raw pcap, show more occurrences of negative inter-arrival times because all flows that are in use while one-time jump happens will experience this time jump. For a clearer picture of the inter-arrival time of benign flows, the distribution of the positive inter-arrival time is shown in figure 4.5 and the negative inter-arrival time in figure 4.4; table 4.5 summarizes the prevalence of inter-arrival times in the different data sets. Additional figures about the distribution of the inter-arrival times are shown in annex A. Further analysis on the origins of the negative inter-arrival times show that only the part of the ISOT data set from the Lawrence Berkeley National Laboratory (LBNL) [45] is impacted by this problem.

	negative IAT ISOT (%)	negative IAT csv (%)	total csv
benign packets	300 (0.002%)	62350 (0.318%)	19,632,070
benign flows	-	62097 (2.93%)	2,114,608
malicious packets	-	0	590,292
malicious flows	-	0	52,659

Tab. 4.5: *Negative inter-arrival times (IAT) in the raw ISOT and extracted csv data set*

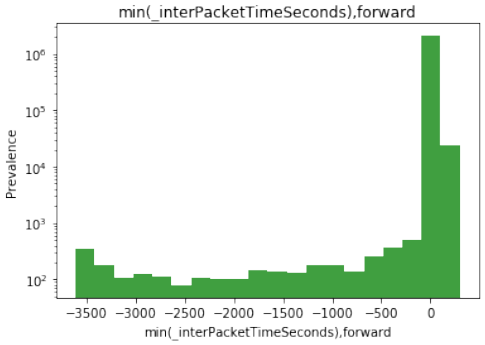

Fig. 4.2: *Histogram of minimum inter-arrival time (in seconds) between packets in benign flows of the extracted data set for the random forest.*

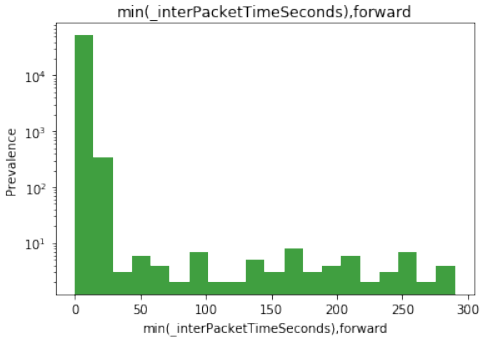

Fig. 4.3: *Histogram of minimum inter-arrival time (in seconds) between packets in malicious flows of the extracted data set for the random forest.*

According to the authors of the data set [10], they replayed all added attacks and a subnet with benign data in which the attacks are embedded in. The raw data from the LBNL was merged with this generated data, but not replayed on a network card. The raw data from the LBNL does however not have these inconsistencies with the negative inter-arrival times, which leads to the assumption that some kind of problem occurred during the merging process. Such a problem might happen because the LBNL data was not replayed on the network card with the other data.

For a more in-depth analysis of the inter-arrival times in the data set, we extracted this feature from the data set and generated a few histogram plots from it. We checked if the inter-

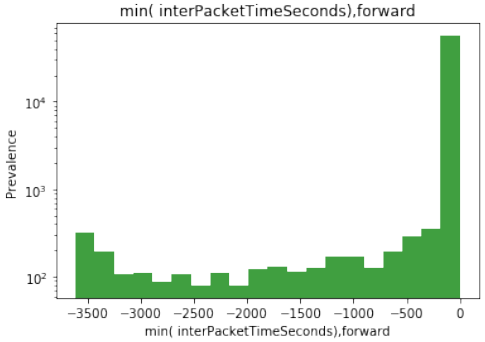

Fig. 4.4: *Histogram of negative minimum inter-arrival time (in seconds) between packets in benign flows of the extracted data set for the random forest.*

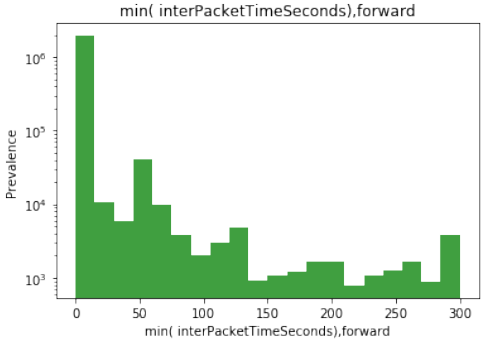

Fig. 4.5: *Histogram of positive minimum inter-arrival time (in seconds) between packets in benign flows of the extracted data set for the random forest.*

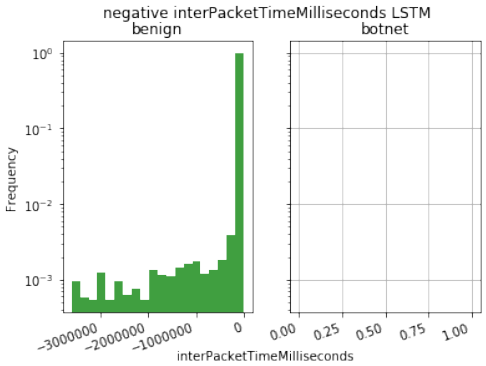

Fig. 4.6: Histogram of negative inter-arrival time between packets (in milliseconds) of the extracted data set for the LSTM autoencoder.

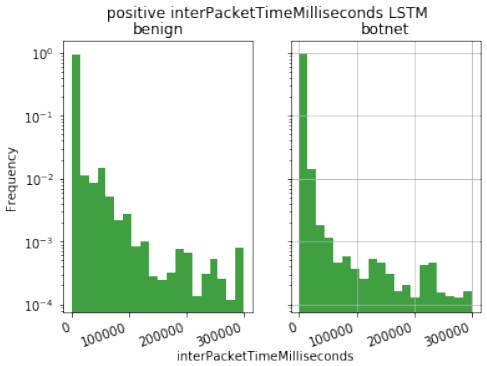

Fig. 4.7: Histogram of positive inter-arrival time between packets (in milliseconds) of the extracted data set for the LSTM autoencoder.

arrival times correlate with the labels, and find that all malicious packets have an inter-arrival time greater than or equal to 0 while some benign packets have negative inter-arrival times. As discussed above, it is in theory possible to find such discrepancies, due to problems in the data set generation. Figure 4.7 shows that for malicious packets, the prevalence of larger inter-arrival times decreases rapidly while for benign packets this decrease is much less pronounced. The malicious packets show a small peak in the interval between 200s and 250s, but for benign packets a similar peak is shifted towards higher inter-arrival times in the area around 250s. Figure 4.7 shows the distribution of all packets with positive inter-arrival times while figure 4.6 shows the distribution of packets in the data set for the LSTM autoencoder with negative inter-arrival times.

4.2 Metrics

In the context of our topic, True Positives (TP) are malicious data instances (packets or flows) correctly identified as being malicious. True Negatives (TN) on the other hand are data packets correctly identified by the algorithm as being non-malicious. False Positives (FP) are non-malicious packets which are mistakenly identified as malicious. False Negatives (FN) are malicious packets which are erroneously classified by the algorithm to be non-malicious.

A multitude of different metrics exist, by which one can compare binary classifications. In our case, we concentrate on the recall, precision, accuracy, F1 score and receiver operating characteristic area under curve (ROC-AUC).

The recall is calculated by dividing the true positives by the sum of true positives and false negatives and shows the probability by which a malicious sample is correctly identified by the machine learning algorithm. It is also sometimes called the true positive rate, sensitivity or hit rate.

$$Recall = \frac{TP}{TP + FN} \tag{4.1}$$

The precision is defined as the number of true positives divided by the sum of true positives and false positives. It therefore shows the number of correctly identified malicious samples in proportion to the total count of samples classified as malicious.

$$Precision = \frac{TP}{TP + FP} \tag{4.2}$$

When dividing the sum of all correctly classified samples by the total number of samples (correctly and falsely classified) the result is called the accuracy.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.3}$$

The F1 score is calculated as the harmonic mean of the precision and recall.

$$F1 = \frac{2TP}{2TP + FP + FN} \tag{4.4}$$

Whether a good choice of parameters can also lead to a good result in terms classifying malware correctly is pictured by the receiver operating characteristic area under curve (ROC-AUC). The recall is being plotted as a function of the false positive rate resulting in a curve. The area under this curve aims to get as near as possible to 1 being the perfect score whilst a value of 0.5 denotes that the classifier under test is no better than a pure random guesser. For the random forest algorithm, we need to use the predict_proba function from SKlearn - in which the *prediction probabilities are calculated as the mean predicted class probabilities of the trees in a forest* [76] - to calculate the ROC-AUC score, since otherwise the score would be computed just on the binary classifier label which is of course nonsensical.

4.3 Implementation

We start with a description of the random forest implementation and then continue with the implementation of the LSTM autoencoder.

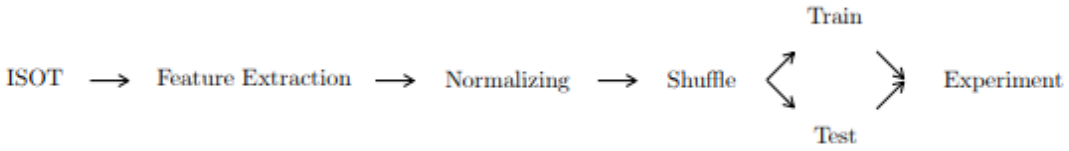

Fig. 4.8: *Flowchart of RF experiment preparation.*

4.3.1 Random Forest

As mentioned in section 4.1, we use the ISOT data set. A random forest is a flow-based algorithm, thus we extract the flows with 43 features from the original data set. The **feature extraction** is described in section 4.1.1 where a list of all features used is found in table 4.2.
NaN values are undefined values or fillers for values that are missing, so that in our case, when for example only the forward part of a flow is available, all computations on the backward part of the flow (such as standard deviation, mean, etc.) are set to NaN. Since the NaNs are, in this case, a representation of a non-existing entry, we can set them to zero. All features are **normalized** with a min-max scaler that was provided by SKlearn [76] as well; it scales each feature to a range between 0 and 1. The scaling is usually necessary in neural network based methods to adjust every feature to the same interval so that the algorithm does not attach special importance to features with larger numbers. For tree-based methods, scaling does not influence results. However, since we scaled the data for the LSTM, we also include this step for the RF classifier. The flows are **shuffled** so the algorithm can't learn from the order of the flows. Afterwards, the data set is **split** into a training (70%) and a testing (30%) part.
The maximum depth and MSL range are then provided as hyperparameters to a search of the best combination of hyperparameters based on a evolutionary genetic algorithm. The properties of this genetic algorithm [77] are the default parameters given by the Python SKlearn [76] implementation:

- Gene mutation probability: 0.10
- Gene cross over probability: 0.5
- Tournament size: 3
- Number of generations: 6

The random forest implementation is also provided by the python 3 package SKlearn [76] with a Number of trees of 1+ Number of features / 2, the tree depth resulting from the genetic

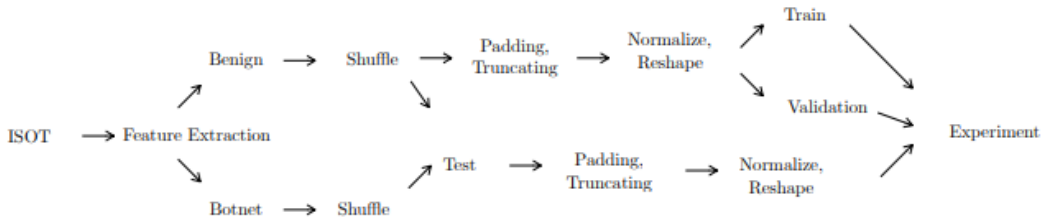

Fig. 4.9: *Flowchart of LSTM experiment preparation.*

algorithm as well as the minimum number of samples per leaf (MSL).

The best random forest selected by the genetic algorithm is then subject to a feature importance check and chosen as the result of the **experiment**. The importance of this feature for classification is based on the *Gini impurity*, which calculated as follows:

$$G = \sum_{i=1}^{n_c} p_i \cdot (1 - p_i) \tag{4.5}$$

n_c denotes the number of classes while p_i denotes the ratio of classes in a branch. The feature importance is then calculated as the mean of the Gini impurity of all branches. The sum of all features is equal to 1. It can also be interpreted as the normalized *total reduction of criterion brought by that feature* [76], as explained in the SKlearn metrics. Figure 4.8 shows all steps taken to prepare the data set for the random forest experiments.

4.3.2 LSTM implementation

The Python 3 programming language is used with the Keras [78] package and tensorflow backend [79] to create the machine learning model and experiments. Keras is a high-level application programming interface (API) that provides a vast array of machine learning building blocks, such as implementations of neural networks, LSTMs, etc.

Section 4.1 describes the ISOT [10] data set used. The **feature extraction** is then described in section 4.1.2. The data set consisting of only benign data is used for training, validation and testing (after being mixed with malicious data) whereas the malicious data is only used for testing. The benign parts for testing are **shuffled** together with the botnet data, while the training and testing sets are shuffled individually. Since not all flows are of the same length, we need to either cut off or add packets to a flow to achieve uniformity. We decide to let each flow contain ten packets (time steps). Thus, we introduce a **padding** for all the flows containing less than ten packets. This padding is created with the Keras [78] pad_sequences command with parameters 'padding' and 'truncating' set to 'post'. It is padded with '2' for all features. This padding is recognized and then masked for the experiments.

Afterwards, the data is **normalized** with a MinMaxScaler [76] using the fit_transform command. Since the MinMaxScaler can only work with 2-dimensional data, but the LSTM autoencoder needs 3 dimensions, the complete data structure is **reshaped** to the correct dimensions: number of samples x time steps x features. Then, a masking layer is introduced to recognize the 0-padding time steps so they wouldn't be trained on. After this normalization, we start with the experiments.

First, the LSTM autoencoder is **trained** on the training data set to recognize benign data. To re-examine whether this process is successful, validation is carried out. The validation data set is however not used to calculate any statistics or key results (such as the threshold) for the model. For the **testing** part, both benign and malicious data is used. Only at this stage to we calculate the threshold, the method is detailed below.

In an autocencoder, the difference between benign and malicious data is portrayed by a reconstruction error. The metric used for the reconstruction error ist the mean squared error. A very good description is found in *Xia et al. 2015* [39]: *"[normal data is] much easier to reconstruct than outliers [,therefore] the larger the difference between the output (i.e. the reconstructed input) and the original input, the more likely it is that the corresponding data is an outlier."* The more the data before encoding and after decoding diverges, the higher this error and subsequently the probability of it being malicious data is. The threshold is the cutoff line between data being classified as benign or malicious: all data with a reconstruction error lower than the threshold is classified as benign, all data with a reconstruction error greater than the threshold as malicious. For an interval from 0 to 1 in steps of 0.01, at each step the F1-score of the respective threshold is calculated. The threshold with the highest F1-score is then selected as the threshold by which the experiment results are classified. In our work, we use the testing data for this.

The LSTM encoding layer consists of three LSTMs provided by Keras [78]. With all layers using sigmoid recurrent activation, in the first two of these layers the return_sequences parameter is set to 'true', whilst the third one is set to 'false'. Between the encoding and decoding layer, we introduce a RepeatVector in order to turn the data passing the auto encoder back into the right dimension for the decoder.

The decoding layer also consists of three LSTMs with sigmoid recurrent activation, yet the return_sequences parameter set to 'true' for all of them. In order to enable the reconstruction of the data set a dense layer is introduced. We select the mean squared error as the loss function for the auto encoder.

Figure 4.9 shows all steps taken to prepare the data set for the long short-term memory experiments, and the code is uploaded to a Github repository [80].

Chapter 5

Experiments

5.1 Experiments with Random Forest

A random forest is an evolution of a decision tree algorithm. As with forests and trees, it is an amplification using many decision trees at the same time and later pooling or averaging their subsequent results. Finally, the random forest classifier's result is the output selected by a majority of the individual random trees. This process is more robust, especially against overfitting of the individual trees to a training set. However, it is a form of supervised learning algorithms, needing labeled training data. In terms of supervised learning algorithms, a random forest is a simple and easy to follow approach. Therefore, it was an initial choice for this thesis. It is described in more detail in Section 2.2.1

First, the shuffled and labeled data set is split into two parts, one of which is used for the training and the other for the testing stage. During the training stage, labeled data is given to the algorithm from which it can learn how to distinguish benign and malicious data. To re-examine whether this learning process was successful, for the testing stage the remaining part of the data set is unlabeled before it is given to the algorithm and a testing is carried out. As the labels are saved separately, we can compare the classification results of the testing with them and see how well the algorithm performs. The data set used in both stages is described in section 4.1.

For the first six experiments with the previous normalization, the maximum minimum number of sample leafs (hereafter MMSL) is set to three, the maximum tree depth is also set to three. However, we discovered that for each of these experiments one feature has over 99% feature importance. We thus gradually eliminate the following features from the feature list: minimum inter arrival time of packets in forward direction, minimum inter arrival time of packets in backward direction, mean inter arrival time of packets in forward direction, maximum inter arrival time of packets in forward direction and standard deviation of inter arrival time

Number	Number of features	MMSL	max depth
1	38	3	3
2	38	4	4
3	38	5	5
4	38	6	6
5	38	7	7
6	38	8	8
7	38	9	9
8	38	10	10
9	38	11	11
10	38	12	12
11	38	13	13
12	38	14	14
13	38	15	15
14	38	16	16
15	38	17	17
16	38	18	18
17	38	19	19
18	38	20	20
19	38	25	25
20	38	30	30
21	38	35	35

Tab. 5.1: *Experiments made with the Random Forest algorithm*

of packets in forward direction. At the end, we are left with 38 different features. All other parameters are kept as is.

These MMSL and max depth values are set as maxima, which means that the algorithm applies values up to the set ones: with a MMSL of three, the algorithm can choose to build random forests consisting of decision trees which have a minimum number of samples per leaf of one, two or three.

For the next seventeen experiments (RF_1 -RF_21), the number of features is kept at 38 but the MMSL and the maximum depth are both incrementally increased from 4 to 20 in each subsequent experiment. This is to check if a higher number of samples per leaf or a higher maximum depth can significantly impact the results of the random forest algorithm. Afterwards, three more experiment are made with a larger spacing, in our case 25, 30 and 35. The aim is to check how the detection performance evolves with different parameters.

An overview of the features used can be found in table 5.2, and the experiments and their parameters can be found in table 5.1. The random forest experiments were run on a computer with a GPU with 2 GB VRAM, 8 core CPU and 16 GB RAM.

Features used	Features removed
sourceTransportPort	min interPacketTime_ms, fwd
destinationTransportPort	mean interPacketTime_ms, fwd
protocolIdentifier	max interPacketTime_ms, fwd
packetTotalCount, fwd	stdev interPacketTime_ms, fwd
octetTotalCount, fwd	min interPacketTime_ms, bwd
min ipTotalLength, fwd	
mean ipTotalLength, fwd	
max ipTotalLength, fwd	
stdev ipTotalLength, fwd	
variance ipTotalLength, fwd	
variance interPacketTime_ms, fwd	
tcpSynTotalCount, fwd	
tcpAckTotalCount, fwd	
tcpFinTotalCount, fwd	
tcpPshTotalCount, fwd	
tcpRstTotalCount, fwd	
tcpUrgTotalCount, fwd	
tcpCwrTotalCount, fwd	
tcpEceTotalCount, fwd	
packetTotalCount, bwd	
octetTotalCount, bwd	
min ipTotalLength, bwd	
mean ipTotalLength, bwd	
max ipTotalLength, bwd	
stdev ipTotalLength, bwd	
variance ipTotalLength, bwd	
mean interPacketTime_ms, bwd	
max interPacketTime_ms, bwd	
stdev interPacketTime_ms, bwd	
variance interPacketTime_ms, bwd	
tcpSynTotalCount, bwd	
tcpAckTotalCount, bwd	
tcpFinTotalCount, bwd	
tcpPshTotalCount, bwd	
tcpRstTotalCount, bwd	
tcpUrgTotalCount, bwd	
tcpCwrTotalCount, bwd	
tcpEceTotalCount, bwd	

Tab. 5.2: *Features used by random forest*

5.2 Experiments with LSTM autoencoder

An introduction to LSTMs is presented in section 2.2.2. Several hyperparameters are relevant for an LSTM autoencoder: batch size, epochs and the number of neurons in each layer. The number of neurons in a layer is interesting to illuminate because too small a number can lead to very poor results but too many can lead to very long durations of the experiments. To generate a reference to which all experiments can be compared to, a first experiment named ISOT_baseline was made with the following hyperparameters, shown in 5.3, originating from a combination of two papers: [35] for using *"approximately 23 neurons per layer"*, which we round to 20, and [34] for using 5 neurons in the last layer.

In a first set we set the neurons in layer 1 and 2 to 100 each, while in layer 3 we change them from 10 to 50 in steps of 10 while batch size and epochs are (identical to the baseline) 50. These experiments are called ISOT_000 to 004.

Name	Neurons Layer 1	Neurons Layer 2	Neurons Layer 3	Batch size	Epochs
ISOT_baseline	20	20	5	50	50
ISOT_000	100	100	10	50	50
ISOT_001	100	100	20	50	50
ISOT_002	100	100	30	50	50
ISOT_003	100	100	40	50	50
ISOT_004	100	100	50	50	50

Tab. 5.3: *Baseline and first set of LSTM autoencoder experiments*

For the next set, the batch size of the two experiments showing the highest AUC (002 and 003), see section 6.2 for all results, was varied from 30 to 120 in changing steps (30, 40, 60, 70, 120) in order to check the impact of the batch size on the AUC. The AUC was calculated by using the auc-function from the SKlearn metrics module, *using the trapezoidal rule* [76]. We skip batch size 50 as we already simulated with these hyperparameters in the previous set. The experiments are called ISOT_010 to 019.

Name	Neurons Layer 1	Neurons Layer 2	Neurons Layer 3	Batch size	Epochs
ISOT_010	100	100	40	30	50
ISOT_011	100	100	30	30	50
ISOT_012	100	100	40	40	50
ISOT_013	100	100	30	40	50
ISOT_014	100	100	40	60	50
ISOT_015	100	100	30	60	50
ISOT_016	100	100	40	70	50
ISOT_017	100	100	30	70	50
ISOT_018	100	100	40	120	50
ISOT_019	100	100	30	120	50

Tab. 5.4: *Second set of LSTM autoencoder experiments*

In a next step, the neurons in layer 1 and 2 are set to 200 each due to the consideration that the more neurons are used the higher the AUC could get as each neuron saves information, therefore the more neurons are available, the more information about the data set can be saved by the algorithm and recalled when evaluating unknown data. At the same time, layer 3 is set to 40 neurons as this number had previously shown consistent results using the same batch sizes of 30, 40, 60, 70 and 120. These series of experiments are called ISOT_020 to 024.

Name	Neurons Layer 1	Neurons Layer 2	Neurons Layer 3	Batch size	Epochs
ISOT_020	200	200	40	30	50
ISOT_021	200	200	40	40	50
ISOT_022	200	200	40	60	50
ISOT_023	200	200	40	70	50
ISOT_024	200	200	50	120	50

Tab. 5.5: *Third set of LSTM autoencoder experiments*

We set a set called ISOT_030 to 037 with the following hyperparameters:

- Batch size 50
- Epochs 50
- Neurons layer 1 200
- Neurons layer 2 200
- Neurons layer 3 ranging from 40 to 110 in steps of 10

This set is based upon the set 000 to 004 while having the doubled number of neurons in layer 1 and 2. As the result from 000 to 002 were not satisfactory in terms of AUC we choose to start

with 40 neurons in layer 3 and going up to 110 in order to check the effect of a higher number of neurons in layer 3 on the AUC:

Name	Neurons Layer 1	Neurons Layer 2	Neurons Layer 3	Batch size	Epochs
ISOT_030	200	200	40	50	50
ISOT_031	200	200	50	50	50
ISOT_032	200	200	60	50	50
ISOT_033	200	200	70	50	50
ISOT_034	200	200	80	50	50
ISOT_035	200	200	90	50	50
ISOT_036	200	200	100	50	50
ISOT_037	200	200	110	50	50

Tab. 5.6: *Fourth set of LSTM autoencoder experiments*

Following the assumption that a higher number of neurons correlates to a higher AUC, we start two further sets raising the neurons of layer 1 and 2 to 300 (ISOT_04*) and 400 (ISOT_05*) each. These two series of experiments are both based on ISOT_02* with a batch size varying from 30–70 in steps of 10 and 120.

Name	Neurons Layer 1	Neurons Layer 2	Neurons Layer 3	Batch size	Epochs
ISOT_040	300	300	40	30	50
ISOT_041	300	300	40	40	50
ISOT_042	300	300	40	50	50
ISOT_043	300	300	40	60	50
ISOT_044	300	300	40	70	50
ISOT_045	300	300	40	120	50
ISOT_050	400	400	40	30	50
ISOT_051	400	400	40	40	50
ISOT_052	400	400	40	500	50
ISOT_053	400	400	40	60	50
ISOT_054	400	400	40	70	50
ISOT_055	400	400	40	120	50

Tab. 5.7: *Fifth and sixth sets of LSTM autoencoder experiments*

To investigate the effect of a changed number of neurons in layer 2 we devise a series of experiments based on ISOT_04*/05* in which we set the neurons of layer 1 to 500 whilst altering the neurons in layer 2 between 250 and 500. Thus we have the following experiments ISOT_06*:

Name	Neurons Layer 1	Neurons Layer 2	Neurons Layer 3	Batch size	Epochs
ISOT_060	500	500	40	30	50
ISOT_061	500	250	40	30	50
ISOT_062	500	500	40	40	50
ISOT_063	500	250	40	40	50
ISOT_064	500	500	40	50	50
ISOT_065	500	250	40	50	50
ISOT_066	500	500	40	60	50
ISOT_067	500	250	40	60	50
ISOT_068	500	500	40	70	50
ISOT_069	500	250	40	70	50
ISOT_0610	500	500	40	120	50
ISOT_0611	500	250	40	120	50

Tab. 5.8: *Seventh of LSTM autoencoder experiments*

As we do not observe big differences between 250 and 500 neurons in layer 2, a new batch (ISOT_07*) is devised for further investigation. For that reason, the neurons in layer 1 are increased to 500 or 1000 with the neurons in layer 2 being reduced to 120 or 250. Layer 3 is kept to 40 neurons; the batch size and epochs are kept at 50.

Name	Neurons Layer 1	Neurons Layer 2	Neurons Layer 3	Batch size	Epochs
ISOT_070	1000	250	40	50	50
ISOT_071	1000	120	40	50	50
ISOT_072	500	250	40	50	50
ISOT_073	500	120	40	50	50

Tab. 5.9: *Eighth set of LSTM autoencoder experiments*

Finally, we want to test the benefits of a very high number of neurons in layer 1 in respect to the increase of computation time needed for such experiments. We set up a series of five experiments to test whether such an increase in neurons is even beneficial. In ISOT_08* we increase the neurons of layer 1 from 1000 to 2000.

Name	Neurons Layer 1	Neurons Layer 2	Neurons Layer 3	Batch size	Epochs
ISOT_080	1000	500	40	50	50
ISOT_081	1200	500	40	50	50
ISOT_082	1500	500	40	50	50
ISOT_083	1800	500	40	50	50
ISOT_084	2000	500	50	50	50

Tab. 5.10: *Ninth sets of LSTM autoencoder experiments*

An overview of all experiments can be found in table 5.11 on the following page. The results will be presented in Chapter 6.2 and discussed in Chapter 7. The LSTM autoencoder experiments were run on a computer with a GPU with 12 GB VRAM, 16 core CPU and 100 GB RAM.

Tab. 5.11: *Experiments made with the LSTM autoencoder algorithm*

Name	Neurons layer 1	Neurons layer 2	Neurons layer 3	Batch size	Epochs
ISOT_000	100	100	10	50	50
ISOT_001	100	100	20	50	50
ISOT_002	100	100	30	50	50
ISOT_003	100	100	40	50	50
ISOT_004	100	100	50	50	50
ISOT_010	100	100	40	30	50
ISOT_011	100	100	30	30	50
ISOT_012	100	100	40	40	50
ISOT_013	100	100	30	40	50
ISOT_014	100	100	40	60	50
ISOT_015	100	100	30	60	50
ISOT_016	100	100	40	70	50
ISOT_017	100	100	30	70	50
ISOT_018	100	100	40	120	50
ISOT_019	100	100	30	120	50
ISOT_020	200	200	40	30	50
ISOT_021	200	200	40	40	50
ISOT_022	200	200	40	60	50
ISOT_023	200	200	40	70	50
ISOT_024	200	200	40	120	50
ISOT_030	200	200	40	50	50
ISOT_031	200	200	50	50	50
ISOT_032	200	200	60	50	50
ISOT_033	200	200	70	50	50
ISOT_034	200	200	80	50	50
ISOT_035	200	200	90	50	50
ISOT_036	200	200	100	50	50
ISOT_037	200	200	110	50	50
ISOT_040	300	300	40	30	50
ISOT_041	300	300	40	40	50
ISOT_042	300	300	40	50	50
ISOT_043	300	300	40	60	50

ISOT_044	300	300	40	70	50
ISOT_045	300	300	40	120	50
ISOT_050	400	400	40	30	50
ISOT_051	400	400	40	40	50
ISOT_052	400	400	40	50	50
ISOT_053	400	400	40	60	50
ISOT_054	400	400	40	70	50
ISOT_055	400	400	40	120	50
ISOT_060	500	500	40	30	50
ISOT_061	500	250	40	30	50
ISOT_062	500	500	40	40	50
ISOT_063	500	250	40	40	50
ISOT_064	500	500	40	50	50
ISOT_065	500	250	40	50	50
ISOT_066	500	500	40	60	50
ISOT_067	500	250	40	60	50
ISOT_068	500	500	40	70	50
ISOT_069	500	250	40	70	50
ISOT_0610	500	500	40	120	50
ISOT_0611	500	250	40	120	50
ISOT_070	1000	250	40	50	50
ISOT_071	1000	120	40	50	50
ISOT_072	500	250	40	50	50
ISOT_073	500	120	40	50	50
ISOT_080	1000	500	40	50	50
ISOT_081	1200	500	40	50	50
ISOT_082	1500	500	40	50	50
ISOT_083	1800	500	40	50	50
ISOT_084	2000	500	40	50	50

Chapter 6

Results

6.1 Results of Random Forest

The random forest experiments always contained both a training as well as a testing stage. For the testing stage the following results are achieved:
Due to previous experiments, albeit based on a different normalization, showing too high of an importance for 5 features, these were removed from the feature list. In experiment 1 the feature importance did not single out one feature. Instead, 16 features were selected by the algorithm, 4 of them reaching an importance from 10 - 22% and the rest showing less than 10%.
These results are shown in Table 6.1.Experiments 1 to 15 selected the maximum possible tree depth. However, when it came to the leaves, only experiment 2, 4, 6, 10 chose the maximum available number. All others showed a lower number of leaves compared to the tree depth.
The random forest algorithm polls the classification result from each tree and selects the final classification with a majority vote. The best results show the best combination of parameters. A general trend towards a better accuracy was found in experiments with a greater tree depth, until a maximum of 22. This also applies for the F1-score, precision and recall. However, the best ROC-AUC is found with a tree depth of 16.
All experiments showed an accuracy from 0.97538- 0.994743, the best experiments matching the results found in literature with the ISOT data set, see table 6.2. In our experiments, the ROC-AUC was between 0.9753 - 0.99825. As this metric could not be found in literature, the results could not be compared. The F1-score was found to be in the range from 0.0 - 0.890467 and could also not be directly compared to literature results. Precision and Recall were also evaluated and were found to be between 0.0 – 0.99168, respectively 0.0 – 0.87730. This wide range observed in recall is due to experiment 6, which consistently showed the worst results also in all other metrics (ROC-AUC 0.97538, F1 0.0, precision 0.0, recall 0.0).
When gradually increasing the maximum tree depth, a higher tree depth generally correlates to

better results in all metrics with experiment 19 scoring best except in ROC-AUC and precision (ROC-AUC 0.99671, F1 0.89046, accuracy 0.99474, recall 0.87730, precision 0.90414). The best precision results (0.99168) were achieved by experiment 2.

An overview of all results of the random forest experiments can be found in table 6.1 on the next page.

Number	Nr of features	MMSL	max depth	best MSL	best depth	ROC AUC	Accuracy	Precision	Recall	F1-score
1	38	3	3	3	3	0.975380	0.975644	0.0	0.0	0.0
2	38	4	4	1	4	0.985687	0.978764	**0.991681**	0.129388	0.221392
3	38	5	5	4	5	0.990202	0.983840	0.953864	0.353813	0.515929
4	38	6	6	3	6	0.992239	0.989531	0.919246	0.626106	0.744002
5	38	7	7	7	7	0.994361	0.990936	0.896552	0.710536	0.791963
6	38	8	8	4	8	0.995250	0.991984	0.895335	0.759662	0.821922
7	38	9	9	8	9	0.996298	0.992445	0.887109	0.790414	0.835945
8	38	10	10	3	10	0.996938	0.992880	0.892109	0.805127	0.846342
9	38	11	11	2	11	0.997575	0.993403	0.903638	0.816305	0.857689
10	38	12	12	4	12	0.997787	0.993671	0.902973	0.829438	0.864578
11	38	13	13	1	13	0.997783	0.993952	0.901801	0.843647	0.871692
12	38	14	14	2	14	0.997732	0.994214	0.902658	0.854823	0.877971
13	38	15	15	2	15	0.997770	0.994448	0.904270	0.863538	0.883398
14	38	16	16	2	16	**0.998253**	0.994528	0.901807	0.870295	0.885677
15	38	17	17	1	17	0.997945	0.994594	0.901130	0.874084	0.887336
16	38	18	18	1	17	0.997945	0.994594	0.901130	0.874084	0.887336
17	38	19	19	1	17	0.997945	0.994594	0.901130	0.874084	0.887336
18	38	20	20	3	19	0.997294	0.994669	0.904453	0.873515	0.888674
19	38	25	25	3	22	0.996716	**0.994743**	0.904147	**0.877304**	**0.890467**
20	38	30	30	3	28	0.993304	0.994706	0.904659	0.874968	0.889516
21	38	35	35	2	22	0.996400	0.994737	0.904578	0.876483	0.890251

Tab. 6.1: *Results of the random forest experiments*

Experiment	Accuracy	Recall	False Alarm Rate	MCC	Std dev F-measure
Al-Jarrah S3	0.999845	0.9942	0.00009	0.9927	0.0031
REPTree	0.999819	0.9930	0.00011	0.9915	0.0028
RTree	0.999816	0.9905	0.00008	0.9913	0.0032
C4.5	0.999811	0.9930	0.00011	0.9911	0.0020
DNN	0.999575	0.9869	0.00029	0.9801	0.0041
SMO	0.999491	0.9962	0.00015	0.9758	0.0033

Tab. 6.2: *Results of experiments found in literature [63] using the ISOT data set.*

6.2 Results of LSTMs

The aim of our experiments was to find a set of hyperparameters for the LSTM autoencoder which show optimal performance. We measure these performances with the Receiver Operating Characteristic - Area Under Curve (ROC-AUC, or simply AUC). As mentioned in section 4.2, the ROC is created by plotting the true-positive rate against the false-positive rate for different thresholds. For varying the threshold, different MSE values were used as decision boundary between benign and malicious. Samples that got a result above the threshold are classified as malicious. The area encompassed under the ROC-curve, the ROC-AUC, describes how well a classifier can detect malicious activity. An AUC of 1 would imply a perfect classifier detecting without any false positives or false negatives, whereas an AUC of 0.5 describes a classifier randomly choosing the classifications. Therefore, the effective usable results range between AUC 0.5 and 1. All further metrics have already been described in chapter 4.2.

Based on a literature review by *Abraham et al. 2018* [11] the maximum achieved ROC-AUC score is 0.99 with the Stratosphere IPS data set [46]. This is set as a benchmark for our experiments to compete against. The experiments are run as described in chapter 5.2.

At first, we performed a separate normalization for benign and botnet data. With this we achieved very good results (ROC-AUC >0.9999 with thresholds between 01. and 0.3), however such a normalization is not correct because it can introduce bias in the classes which influences the detection performance. In addition, we detected negative inter packet time values in the benign data set (but not in the botnet data). These artifacts are outliers only in the benign data an therefore cause a large shift in the normalized values. It is thus likely that the previously reported good results were only caused by the wrong normalization in combination with outlier from the artifacts. Regrettably, this mistake was undiscovered for a while. As soon as we became aware of this mistake, the normalization was repeated, this time with botnet and benign data using the same normalization, and the experiments were repeated. These new results cannot match the previous ones. We notice that the best AUC is achieved by Experiment ISOT_045 at 0.637. All other experiments (except one) achieve an AUC of at least 0.507. The threshold of

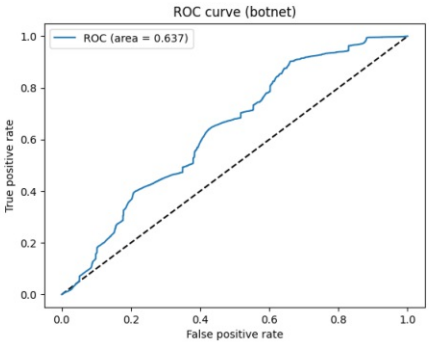

Fig. 6.1: *ROC curve of the best performing experiment.*

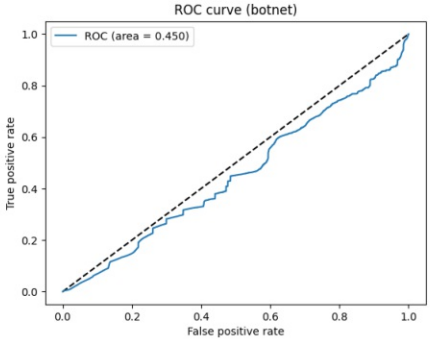

Fig. 6.2: *ROC curve of the worst performing experiment.*

all simulations is set at 0, because this gave the best results for the classifier. The ROC-curve for ISOT_045 ist shown in 6.1 and for the worst performing experiment (ISOT_011) in 6.2. All results discussed in this section are shown in table 6.3 in a concise manner. In the appendix B, the MSE for the TCP flags, benign or malicious, are found. Additionally, the reconstruction error and threshold with its F1-score with the scatterplot of benign and malicious packets are also in the annex. The reconstruction error is calculated as the mean square of the predicted data-points subtracted from the real data-points.

Tab. 6.3: *Results of LSTM autoencoder experiments*

name	threshold	AUC	accuracy	precision	recall	f1-score
ISOT_000	0.0	0.546989	0.723802	0.723802	1.0	0.839774
ISOT_001	0.0	0.566206	0.723802	0.723802	1.0	0.839774
ISOT_002	0.0	0.601628	0.723802	0.723802	1.0	0.839774
ISOT_003	0.0	0.583936	0.723802	0.723802	1.0	0.839774
ISOT_004	0.0	0.527389	0.723802	0.723802	1.0	0.839774
ISOT_010	0.0	0.569080	0.723802	0.723802	1.0	0.839774
ISOT_011	0.0	0.449944	0.723802	0.723802	1.0	0.839774
ISOT_012	0.0	0.549964	0.723802	0.723802	1.0	0.839774
ISOT_013	0.0	0.577193	0.723802	0.723802	1.0	0.839774
ISOT_014	0.0	0.607330	0.723802	0.723802	1.0	0.839774
ISOT_015	0.0	0.542848	0.723802	0.723802	1.0	0.839774
ISOT_016	0.0	0.540393	0.723802	0.723802	1.0	0.839774
ISOT_017	0.0	0.544407	0.723802	0.723802	1.0	0.839774
ISOT_018	0.0	0.599773	0.723802	0.723802	1.0	0.839774
ISOT_019	0.0	0.580644	0.723802	0.723802	1.0	0.839774
ISOT_020	0.0	0.568908	0.723802	0.723802	1.0	0.839774
ISOT_021	0.0	0.507493	0.723802	0.723802	1.0	0.839774
ISOT_022	0.0	0.520279	0.723802	0.723802	1.0	0.839774
ISOT_023	0.0	0.530840	0.723802	0.723802	1.0	0.839774
ISOT_024	0.0	0.549771	0.723802	0.723802	1.0	0.839774
ISOT_030	0.0	0.542477	0.723802	0.723802	1.0	0.839774
ISOT_031	0.0	0.605210	0.723802	0.723802	1.0	0.839774
ISOT_032	0.0	0.568096	0.723802	0.723802	1.0	0.839774
ISOT_033	0.0	0.549595	0.723802	0.723802	1.0	0.839774
ISOT_034	0.0	0.515042	0.723802	0.723802	1.0	0.839774
ISOT_035	0.0	0.570335	0.723802	0.723802	1.0	0.839774
ISOT_036	0.0	0.535696	0.723802	0.723802	1.0	0.839774
ISOT_037	0.0	0.527929	0.723802	0.723802	1.0	0.839774
ISOT_040	0.0	0.578619	0.723802	0.723802	1.0	0.839774
ISOT_041	0.0	0.557202	0.723802	0.723802	1.0	0.839774
ISOT_042	0.0	0.605636	0.723802	0.723802	1.0	0.839774
ISOT_043	0.0	0.553386	0.723802	0.723802	1.0	0.839774

Tab. 6.3: *Results of LSTM autoencoder experiments*

name	threshold	AUC	accuracy	precision	recall	f1-score
ISOT_044	0.0	0.547465	0.723802	0.723802	1.0	0.839774
ISOT_045	0.0	**0.637169**	0.723802	0.723802	1.0	0.839774
ISOT_050	0.0	0.557609	0.723802	0.723802	1.0	0.839774
ISOT_051	0.0	0.595246	0.723802	0.723802	1.0	0.839774
ISOT_052	0.0	0.542238	0.723802	0.723802	1.0	0.839774
ISOT_053	0.0	0.579823	0.723802	0.723802	1.0	0.839774
ISOT_054	0.0	0.586144	0.723802	0.723802	1.0	0.839774
ISOT_055	0.0	0.515051	0.723802	0.723802	1.0	0.839774
ISOT_060	0.0	0.522367	0.723802	0.723802	1.0	0.839774
ISOT_061	0.0	0.540737	0.723802	0.723802	1.0	0.839774
ISOT_062	0.0	0.597789	0.723802	0.723802	1.0	0.839774
ISOT_063	0.0	0.544951	0.723802	0.723802	1.0	0.839774
ISOT_064	0.0	0.554115	0.723802	0.723802	1.0	0.839774
ISOT_065	0.0	0.508448	0.723802	0.723802	1.0	0.839774
ISOT_066	0.0	0.557912	0.723802	0.723802	1.0	0.839774
ISOT_067	0.0	0.552052	0.723802	0.723802	1.0	0.839774
ISOT_068	0.0	0.554237	0.723802	0.723802	1.0	0.839774
ISOT_069	0.0	0.505602	0.723802	0.723802	1.0	0.839774
ISOT_0610	0.0	0.551995	0.723802	0.723802	1.0	0.839774
ISOT_0611	0.0	0.511414	0.723802	0.723802	1.0	0.839774
ISOT_070	0.0	0.593541	0.723802	0.723802	1.0	0.839774
ISOT_071	0.0	0.512012	0.723802	0.723802	1.0	0.839774
ISOT_072	0.0	0.524455	0.723802	0.723802	1.0	0.839774
ISOT_073	0.0	0.556778	0.723802	0.723802	1.0	0.839774
ISOT_080	0.0	0.573426	0.723802	0.723802	1.0	0.839774
ISOT_081	0.0	0.593448	0.723802	0.723802	1.0	0.839774
ISOT_082	0.0	0.608244	0.723802	0.723802	1.0	0.839774
ISOT_083	0.0	0.541571	0.723802	0.723802	1.0	0.839774
ISOT_084	0.0	0.561229	0.723802	0.723802	1.0	0.839774

Chapter 7

Discussion

7.1 Interpretation of results of Random Forest experiments

We shall first concentrate on interpreting the results of our random forest experiments: overall, the random forest achieves solid results. Our lowest ROC-AUC is at 0.97538 while the highest achieves 0.99825. The lowest ROC-AUC is reached by RF_1, having a maximum tree depth of 3 whereas the best result is achieved by RF_14 with a maximum tree depth of 16. It has to be mentioned that experiment RF_1 is definitely an outlier compared to all other experiments; the next better experiment RF_2 (with a tree depth of 4) already scores a ROC-AUC of 0.98568. We assume this being due to the overall trend that shows a correlation of better results with a higher tree depth. Analogous to the popular game called *Twenty Questions*, where players have to guess a term by only asking questions answerable with "yes" or "no" and often a certain minimum number of questions are necessary for the players to find the correct answer, it is also possible that a certain minimum tree depth is necessary for a good classification.

The recall is found to be in an interval between 0.0 and 0.87730 and follows the trend set by the ROC-AUC. RF_1 shows the worst results and RF_19 the best, the same situation is to be found concerning the F1 score which scores between 0.0 and 0.89046. These metrics are consistent within the picture that a higher tree depth leads to better results.

When looking at the minimum samples per leaf (MSL), we see that the optimizer, except in RF_1, RF_5 and RF_7, never came close to maxing out this parameter.

When only looking at the precision, one might be led to the misassumption that experiment RF_2 performs well. However, when also considering ROC-AUC, recall and F1 score, it is obvious that this is not the fact. This is, with the ROC-AUC, the only metrics in which RF_19 is not showing the best results. The overall precision is high, but there ist definitely room for improvement.

High precision and accuracy imply that only a very small amount of benign flows are classified

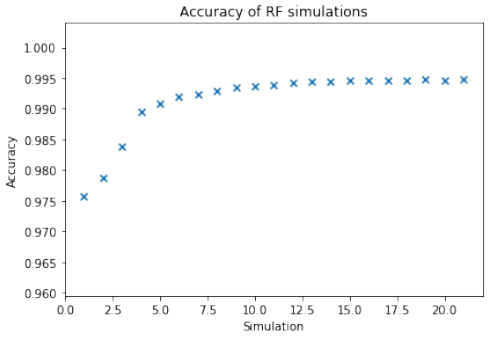

Fig. 7.1: *Accuracy of all RF experiments.*

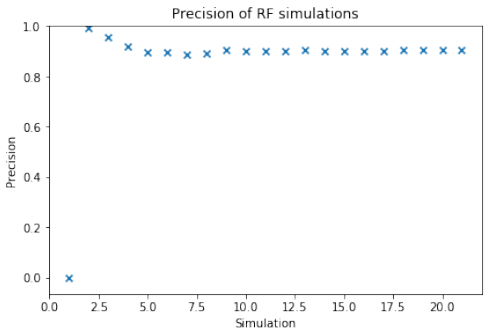

Fig. 7.2: *Precision of all RF experiments.*

as being malicious. This does not mean, however, that the algorithm correctly differentiates between benign and malicious flows when relying only on the precision. One must also consider the recall, which describes how well an algorithm correctly identifies malicious flows in relation to the total number of malicious flows. Our results show an overall smaller recall, implying that quite an amount of malicious flows are classified to be benign in our experiments. This is exemplified in experiments RF_1 and RF_2 - both have high accuracy and RF_2 has the best precision of all experiments - but their recall and F1-score is abysmal. However, considering the greater danger presented by a successful malware attack on a target it is crucial to detect as many malicious activities as possible. Therefore, a high recall is desired, even at the expense of a lower precision.

In previous experiments (see section 5.1) we had to remove five features until the feature importance was distributed among more than one feature. All of them involve the inter arrival time between packets, four out of five in the forwards direction and one in the backwards direction. When looking at the individual values of these features, we see that, for three of these features (mean, max, std dev), in the whole extracted data set there are only two distinct intervals, one of which corresponds to the malicious flows, the other one belonging to benign flows. Since for the data set generation a benign and a malicious data set were combined that originated from different networks, it is quite likely that different inter-arrival times occurred in the different data sets. Nevertheless, between the ISOT and LBNL data sets no direct differences or dependencies can be observed.

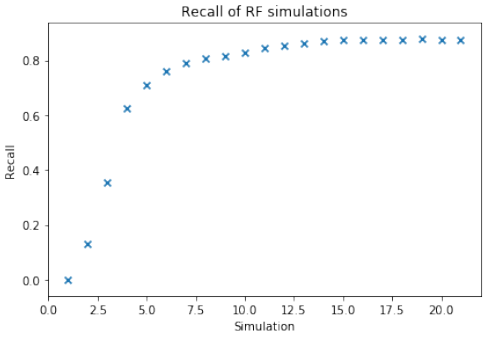

Fig. 7.3: *Recall of all RF experiments.*

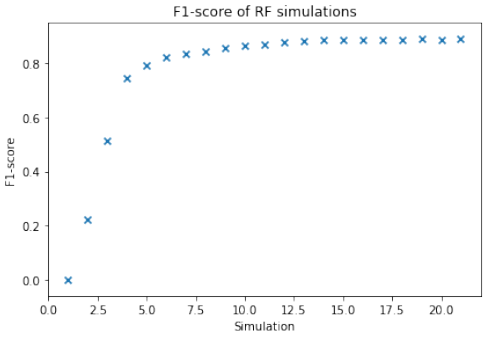

Fig. 7.4: *F1 score of all RF experiments.*

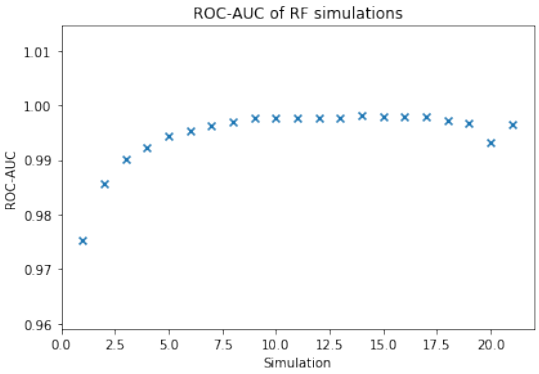

Fig. 7.5: *ROC-AUC of all RF experiments.*

7.2 Interpretation of results of LSTM autoencoder experiments

According to our results, the batch size does not appear to have any influence on the ability of the algorithm to classify our data as shown by experiments ISOT_01*, 02*, 04*, 05*, 06*.

Also, there is no greater benefit in increasing the neurons in layer 1 and 2 to more than 400 for the results do not get better whereas the computational time increases tremendously. The neurons in layer 3 are best to be kept much smaller than the number of timesteps time the number of per-packet features, with an absolute maximum of 100 and 30 or 40 showing better results than the rest.

When assessing the mean squared error (MSE) for each feature and each time step, we can conclude that most, if not all, of the TCP flags do not impact the results of our experiments. In benign as well as malicious data, the MSE for the CWR, ECE, FIN, NS, PSH, RST, SYN and URG is always 0. A likely explanation for this observation is that these flags are almost never used in the data set and do not bring an advantage. The MSE for the remaining flag, ACK, can be greater than 0, in the case of benign as well as in the case of malicious data. This is not a surprise as ACK is one of the most used flags in the TCP protocol and is set in all packets of the TCP conversation (with the exception of connection setup and teardown).

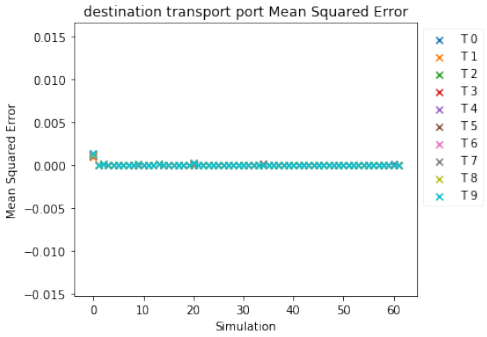

Fig. 7.6: *Mean squared error of the benign destination transport port at each timestep and experiment.*

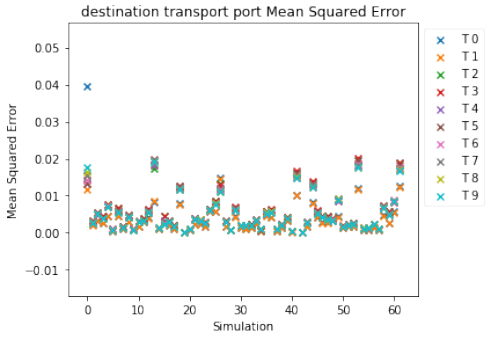

Fig. 7.7: *Mean squared error of the malicious destination transport port at each timestep and experiment.*

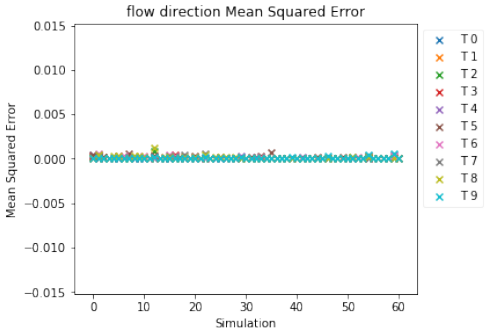

Fig. 7.8: *Mean squared error of the benign flow direction at each timestep and experiment.*

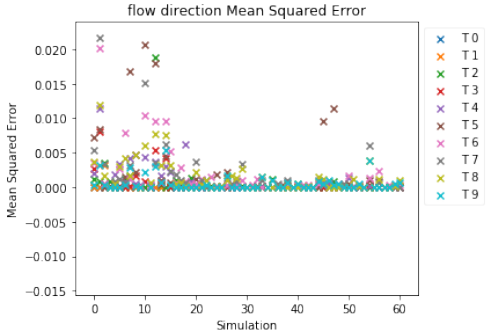

Fig. 7.9: *Mean squared error of the malicious flow direction at each timestep and experiment.*

Contrary to the flags, we can observe a slightly different behavior for the other features, namely destination transport port (figure 7.12), flow direction (figure 7.13), inter arrival time and IP total length. We can also see that timestep 2 has the lowest mean squared error, implying that it is the easiest to reconstruct. The IP total length shows an MSE smaller than 0.005 in all cases, see figure 7.11. Overall, we observe that – except for the inter arrival time – time step 9 hardly ever shows an MSE other than 0 which leads to the assumption that time step 9 might have been padded often. Interestingly, the mean squared error of the malicious samples for the destination transport port (Figure 7.12) is for all LSTM autoencoder experiments always less than 0.04. This means that the LSTM autoencoder can reconstruct the malicious samples quite well. This can be the result of an over-simplification on the part of the autoencoder and is a pitfall of this class of neural networks. For a deeper dive into this fascinating topic, we refer to [61]. Also, as *Iglesias, Hartl et al.* [81] describe, the measure of "outlierness" is difficult for algorithms leveraging local feature space interpretation. Some additional plots can be found in the appendix B.

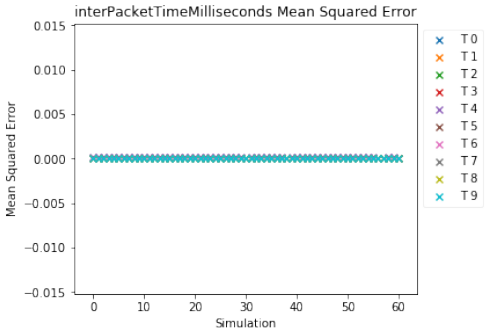

Fig. 7.10: *Mean squared error of the benign interarrival time at each timestep and experiment.*

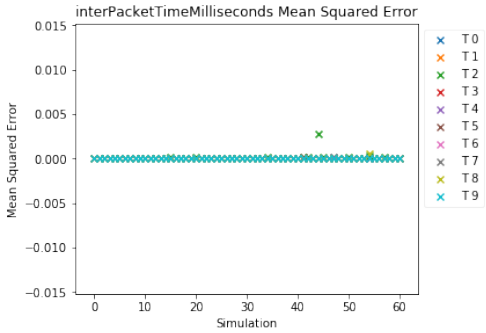

Fig. 7.11: *Mean squared error of the malicious interarrival time at each timestep and experiment.*

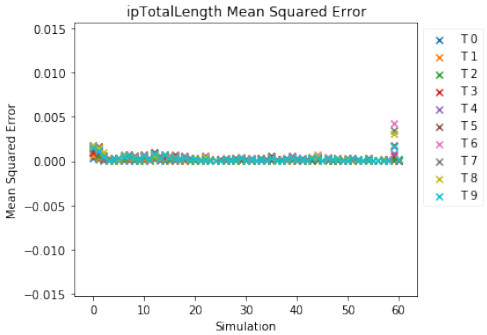

Fig. 7.12: *Mean squared error of the benign IP total length at each timestep and experiment.*

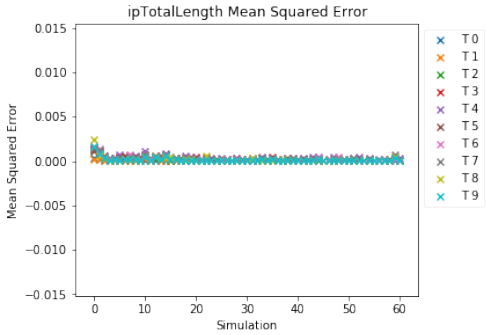

Fig. 7.13: *Mean squared error of the malicious IP total length at each timestep and experiment.*

7.3 Major findings

We use similar features for both machine learning algorithms but they still differ because of their different approach. For this reason, we cannot compare them directly but look at their performance under equitable conditions. Despite its simpler design, the random forest far outperforms the results of the more recent LSTM autoencoder. We observe the general trend that a higher tree depth is beneficial for the performance of the random forest up to a certain point (tree depth 19 in our experiments), after which the performance stabilizes on a high level with the MSL always staying at a low number between one to six. We also observe the great influence the normalization has on the algorithm: when normalizing (by mistake) benign and botnet data separately we achieve very good results and the algorithm can distinguish benign from botnet activity. In our case the extremely strong effect could have been caused by the combination of the wrong normalization together with outliers in the benign data. The outliers were caused by artifacts in the data set, that have not been reported by previous users of the data set. In contrast, with a correct normalization, this did not work. The LSTM autoencoder shows much worse results and thus cannot make use of its theoretical advantage: to detect malicious activity after a few packets. Also, it should be noted that choosing different values for the hyperpa-

rameters do not have a great influence on the performance in our experiments; however, this observation is only valid for the chosen data set and might not be applicable for different data. Irrespective of the hyperparameters, the TCP flags seem not to have much influence on the classification in both LSTM autoencoder and random forest. The distribution of the inter-arrival time for benign samples is distinct from that of the malicious samples: benign samples have a much greater proportion of higher inter-arrival times, a behavior that is not indicated by the malicious samples. Additionally, the raw ISOT data set displayed some negative inter-arrival times, which is of course not possible and probably caused by the merging of different data sets when the ISOT data was generated. Considering these facts, one must be cautious when using this feature for training with this data set. The negative interarrival times are only found in the parts imported from the Lawrence Berkeley National Laboratory (LBNL) [45]. However, the "original" LBNL data set does not have these artifacts in the inter-arrival time and it is therefore likely that the error occurred during the merging.

Chapter 8

Conclusion

Our aim was to check whether a LSTM autoencoder detects a botnet infection in a network more accurately than a random forest. We can assert that the random forest algorithm detects malicious data much better than the LSTM autoencoder, the best result shown by a LSTM autoencoder being 0.637169 in comparison to 0.999481 achieved by the RF.
In respect to the reliability to detect malicious data with a ROC-AUC greater than 0.99 we can also confirm that the random forest achieves this goal.
The different hyperparameters of the LSTM autoencoder do not have a big influence on the detection probability of our data set. This is contrary to the random forest, where we can definitely conclude that choosing the correct tree depth is crucial for a good classification. The main advantage of the random forest algorithm is that it is not very resource intensive. From our observation, a single LSTM autoencoder experiment can take up 15 hours while the RF experiments never took longer than two hours. Additionally, to run such intense experiments as a LSTM autoencoder, extremely capable hardware is needed, whereas a RF can be performed by much less capable hardware. Since an LSTM autoencoder uses per-packet features, the preprocessed data set is much larger than that of a random forest. Compounded by the fact that due to gradient descent (epochs) the whole data set had to be iterated multiple times, the resource consumption is much higher for a LSTM autoencoder.
However, our findings are limited by the quality of the data set. We used a data set generated by ISOT [10], but the interarrival times of the flows seem to be influenced by the generation process so that we had to remove 5 features in order to get a correctly performing RF experiment. It would be desirable to check whether similar results can be achieved when using other data sets. Furthermore, a consensus on a standard set of metrics would be of great advantage to compare results. Future research could also concentrate on the explainability of the different algorithms, in particular the LSTM autoencoder. Additionally, further investigation into whether the LSTM autoencoder can correctly classify flows with more than 10 packets per flow or is prone to over-

simplification could be worthwhile.

We conclude, that the RF is much more capable than the LSTM autoencoder considering the requirements and constraints of our data set.

Appendix A

Figures for Dataset and Random Forest

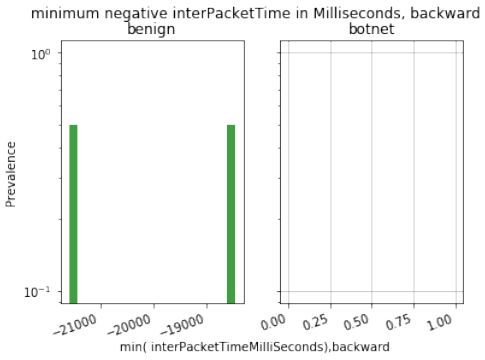

Fig. 1.1: *Minimum negative interarrival time in milliseconds in backward direction.*

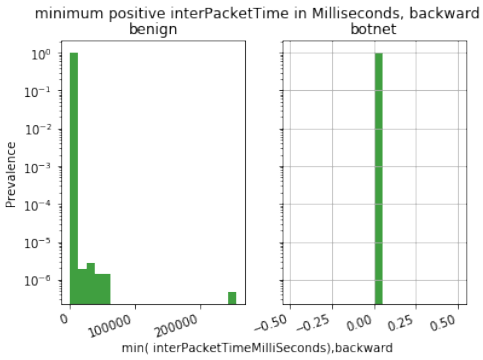

Fig. 1.2: *Minimum positive interarrival time in milliseconds in backward direction.*

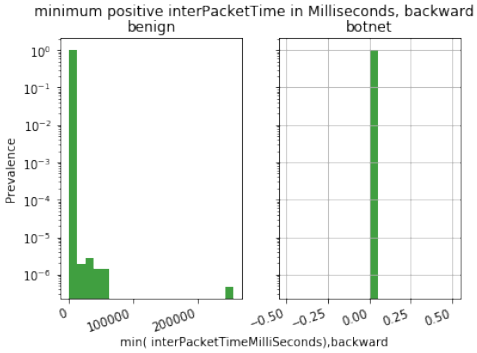

Fig. 1.3: *Normalized minimum interarrival time in backward direction.*

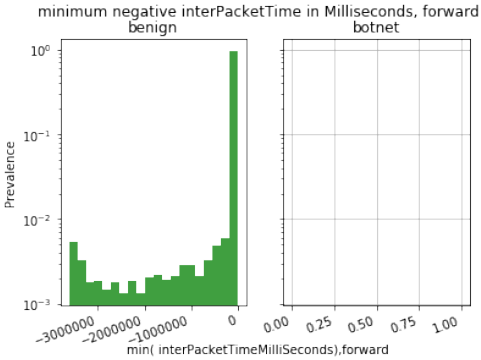

Fig. 1.4: *Minimum negative interarrival time in milliseconds in forward direction.*

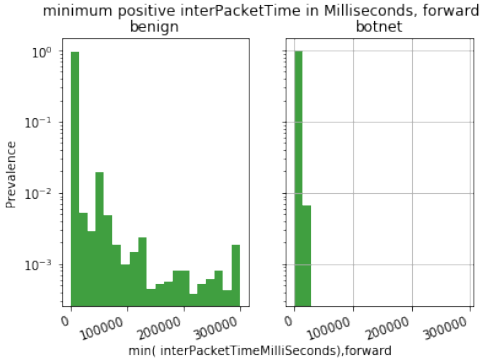

Fig. 1.5: *Minimum positive interarrival time in milliseconds in forward direction.*

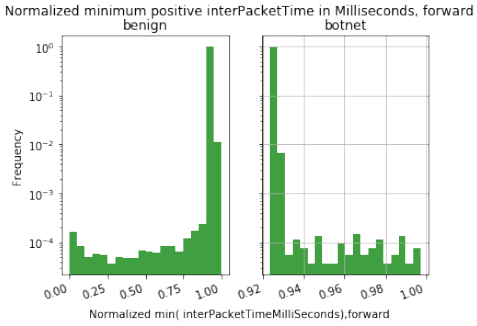

Fig. 1.6: *Normalized minimum interarrival time in forward direction.*

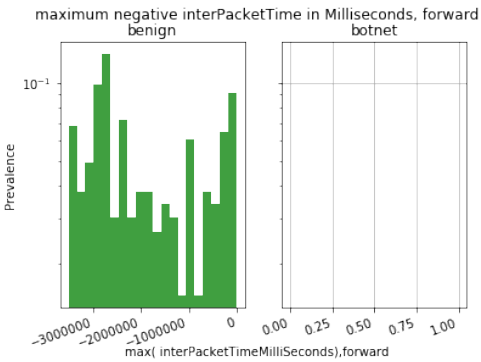

Fig. 1.7: *Maximum negative interarrival time in milliseconds in forward direction.*

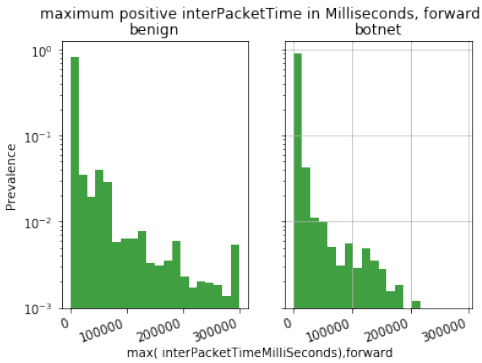

Fig. 1.8: *Maximum positive interarrival time in milliseconds in forward direction.*

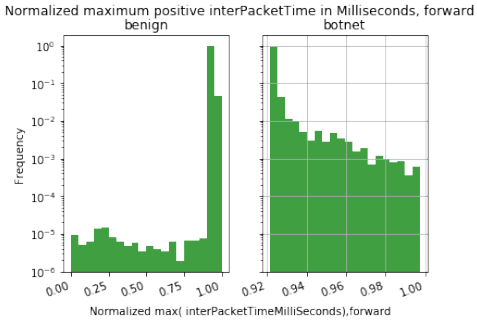

Fig. 1.9: *Normalized maximum interarrival time in forward direction.*

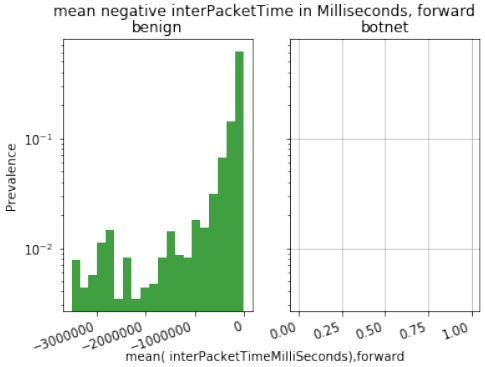

Fig. 1.10: *Mean negative interarrival time in milliseconds in forward direction.*

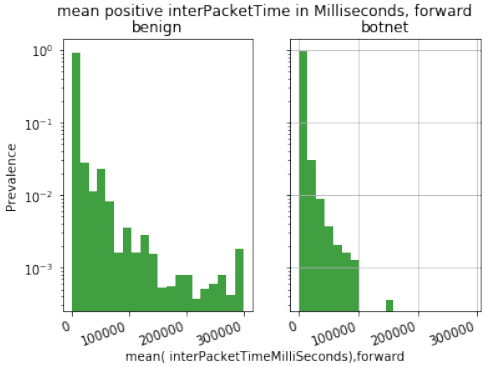

Fig. 1.11: *Mean positive interarrival time in milliseconds in forward direction.*

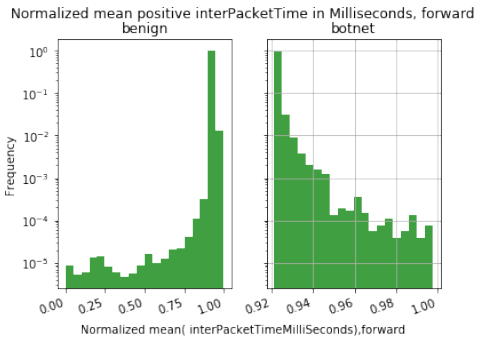

Fig. 1.12: *Normalized mean interarrival time in forward direction.*

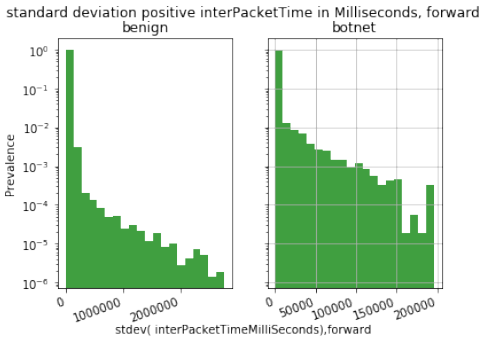

Fig. 1.13: *Standard deviation interarrival time in milliseconds in forward direction.*

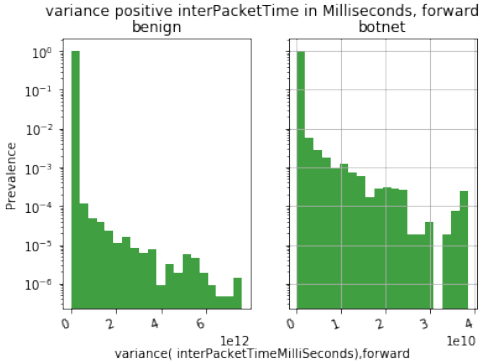

Fig. 1.14: *Variance interarrival time in milliseconds in forward direction.*

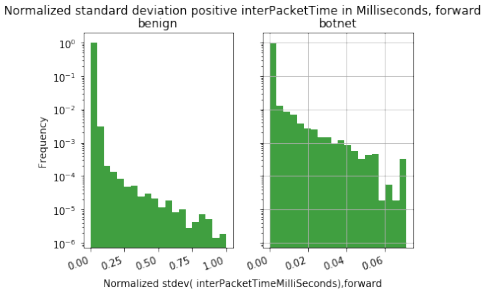

Fig. 1.15: *Normalized standard deviation interarrival time in forward direction.*

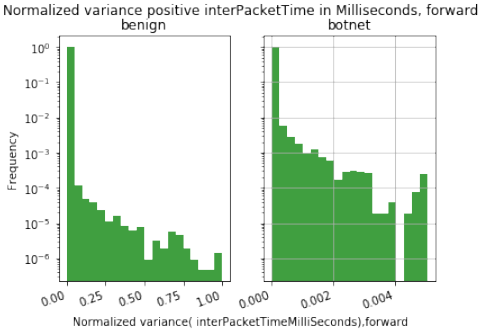

Fig. 1.16: *Normalized variance interarrival time in forward direction.*

Appendix B

Figures for LSTM

The following figures show the mean squared error of all reconstructed benign and malicious flows.

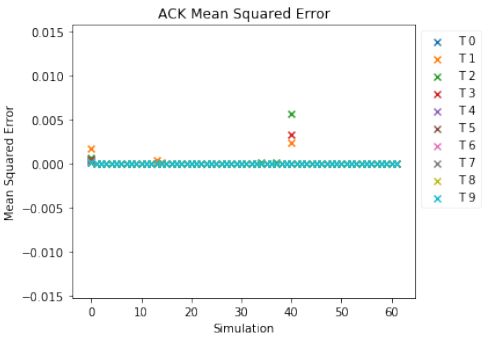

Fig. 2.1: *ACK flag in benign flows.*

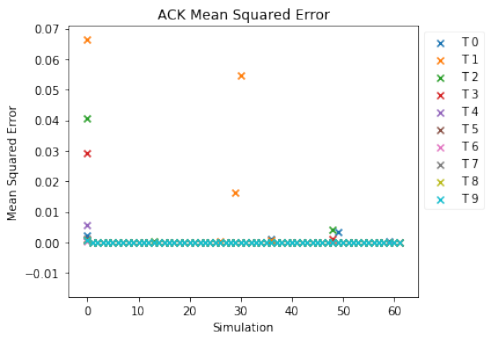

Fig. 2.2: *ACK flag in malicious flows.*

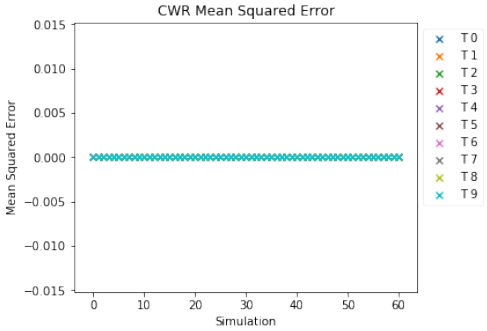

Fig. 2.3: *CWR flag in benign flows.*

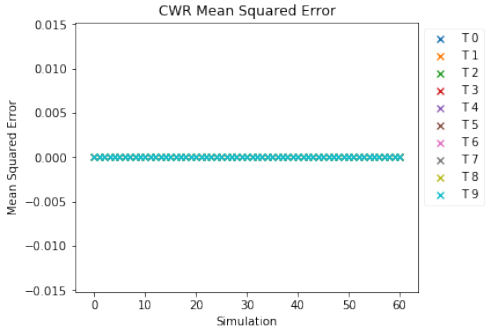

Fig. 2.4: *CWR flag in malicious flows.*

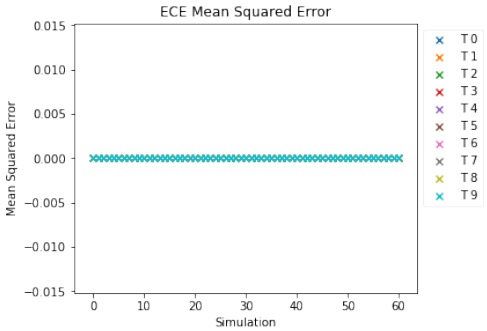

Fig. 2.5: *ECE flag in benign flows.*

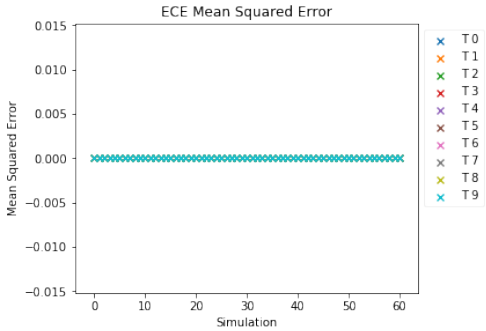

Fig. 2.6: *ECE flag in malicious flows.*

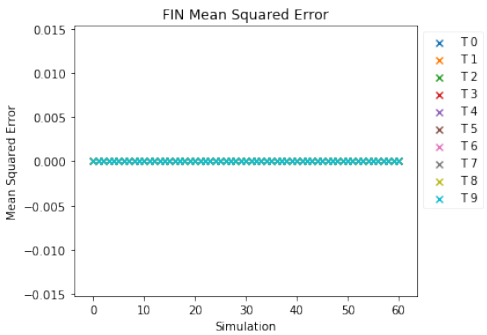

Fig. 2.7: *FIN flag in benign flows.*

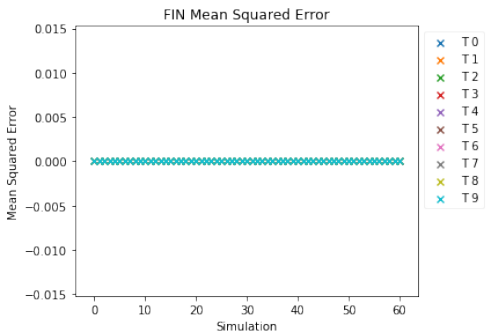

Fig. 2.8: *FIN flag in malicious flows.*

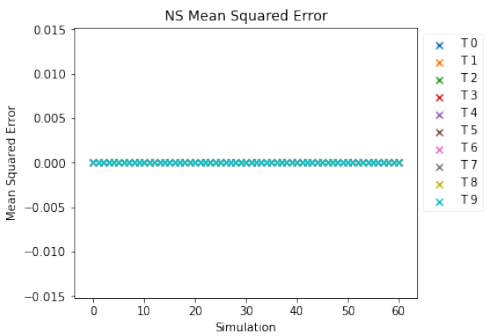

Fig. 2.9: *NS flag in benign flows.*

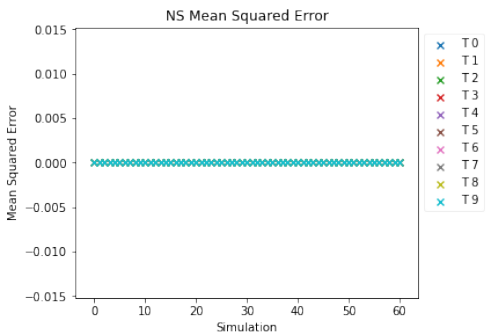

Fig. 2.10: *NS flag in malicious flows.*

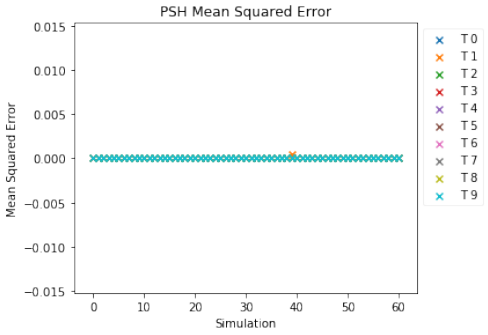

Fig. 2.11: *PSH flag in benign flows.*

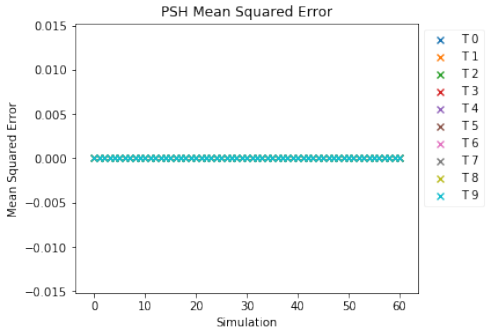

Fig. 2.12: *PSH flag in malicious flows.*

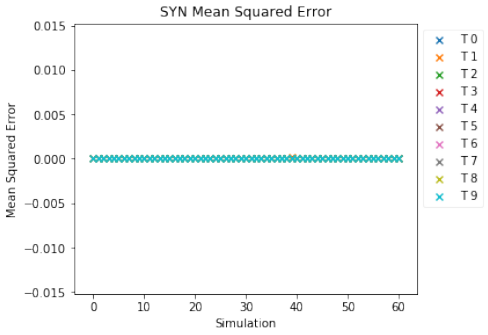

Fig. 2.13: *SYN flag in benign flows.*

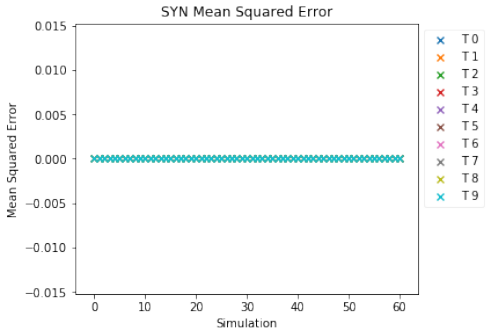

Fig. 2.14: *SYN flag in malicious flows.*

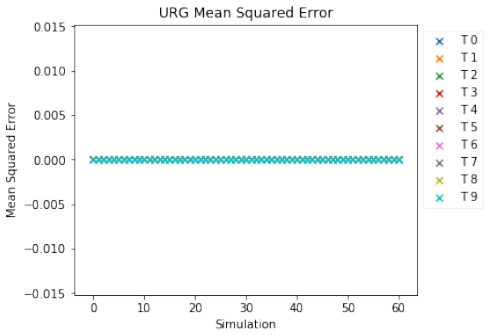

Fig. 2.15: *URG flag in benign flows.*

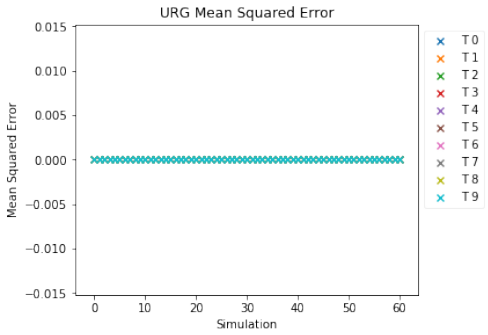

Fig. 2.16: *URG flag in malicious flows.*

Fig. 2.17: *Reconstruction error and threshold of experiment ISOTbaseline.*

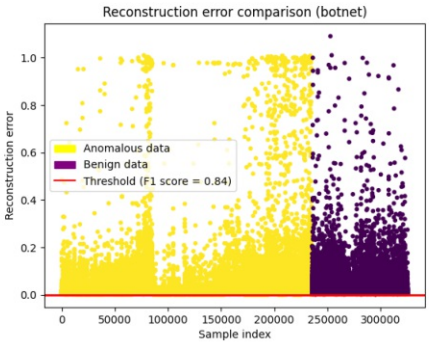

Fig. 2.18: *Reconstruction error and threshold of experiment ISOT_000.*

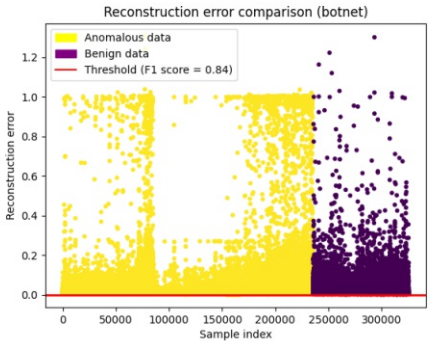

Fig. 2.19: *Reconstruction error and threshold of experiment ISOT_001.*

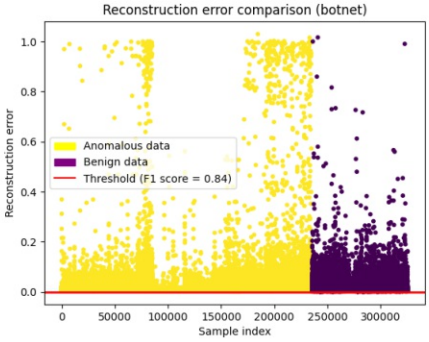

Fig. 2.20: *Reconstruction error and threshold of experiment ISOT_002.*

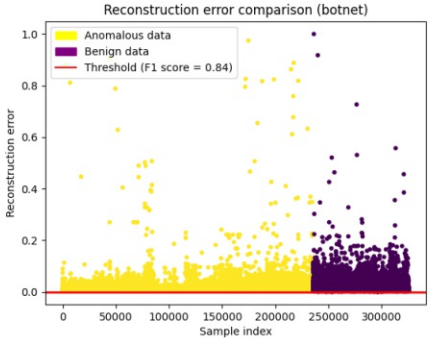

Fig. 2.21: *Reconstruction error and threshold of experiment ISOT_003.*

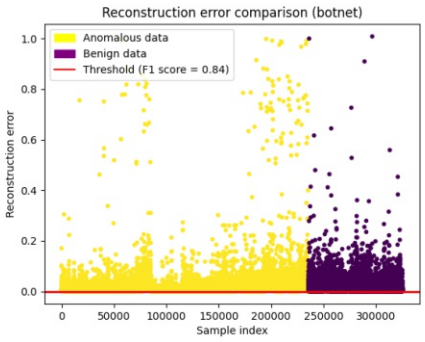

Fig. 2.22: *Reconstruction error and threshold of experiment ISOT_004.*

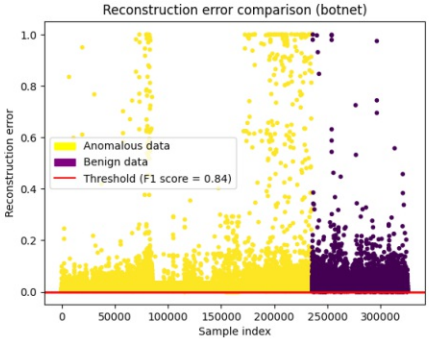

Fig. 2.23: *Reconstruction error and threshold of experiment ISOT_010.*

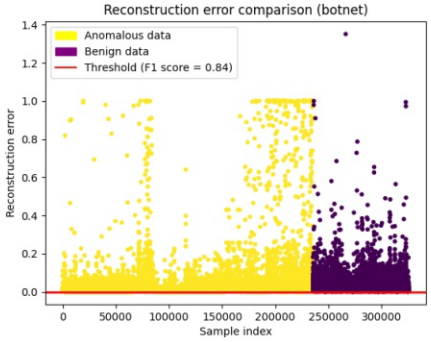

Fig. 2.24: *Reconstruction error and threshold of experiment ISOT_011.*

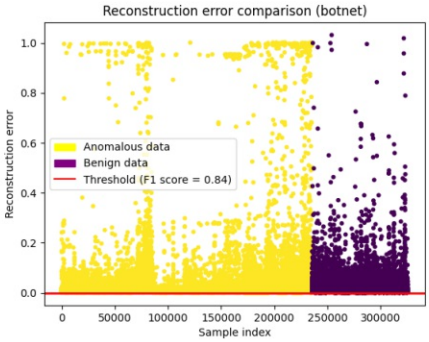

Fig. 2.25: *Reconstruction error and threshold of experiment ISOT_012.*

Fig. 2.26: *Reconstruction error and threshold of experiment ISOT_013.*

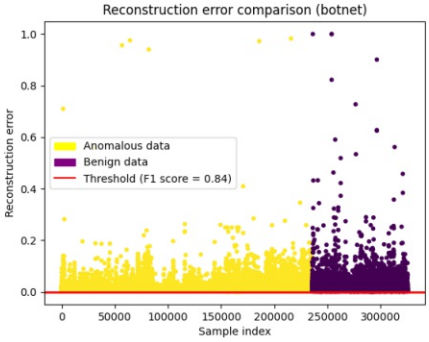

Fig. 2.27: *Reconstruction error and threshold of experiment ISOT_014.*

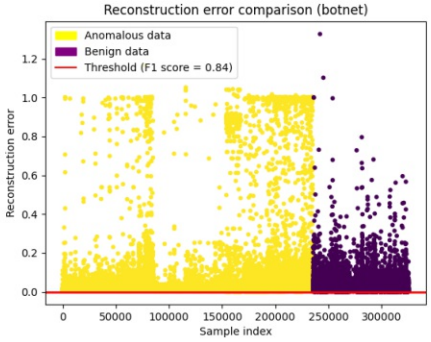

Fig. 2.28: *Reconstruction error and threshold of experiment ISOT_015.*

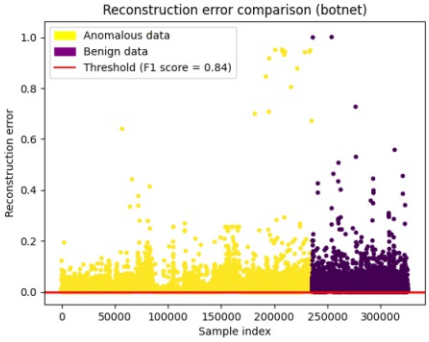

Fig. 2.29: *Reconstruction error and threshold of experiment ISOT_016.*

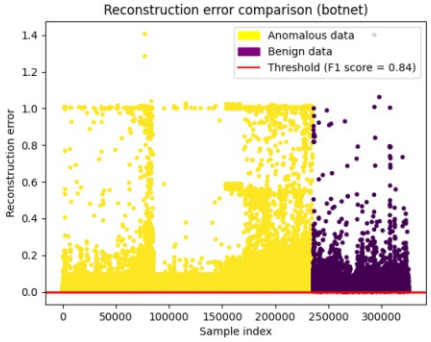

Fig. 2.30: *Reconstruction error and threshold of experiment ISOT_017.*

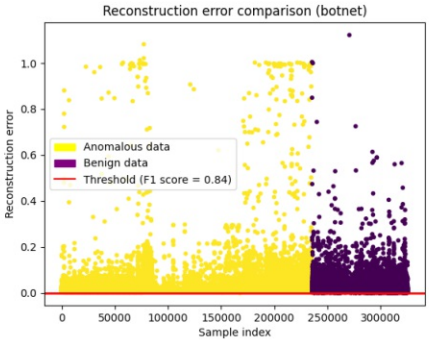

Fig. 2.31: *Reconstruction error and threshold of experiment ISOT_018.*

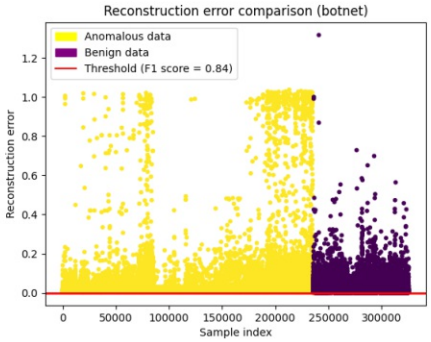

Fig. 2.32: *Reconstruction error and threshold of experiment ISOT_019.*

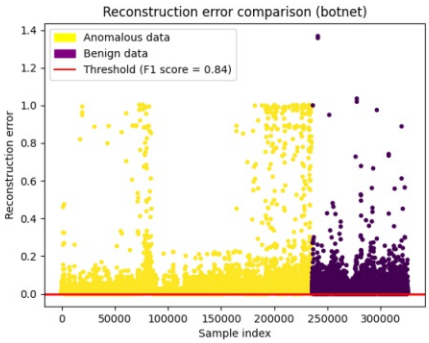

Fig. 2.33: *Reconstruction error and threshold of experiment ISOT_020.*

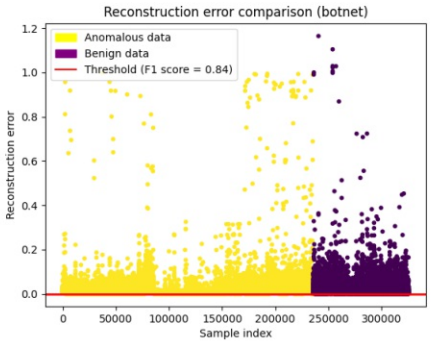

Fig. 2.34: *Reconstruction error and threshold of experiment ISOT_021.*

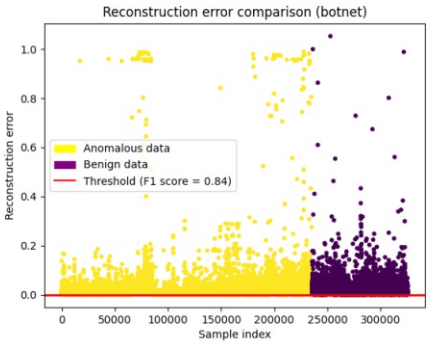

Fig. 2.35: *Reconstruction error and threshold of experiment ISOT_022.*

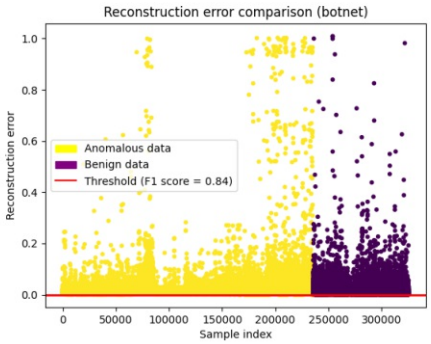

Fig. 2.36: *Reconstruction error and threshold of experiment ISOT_023.*

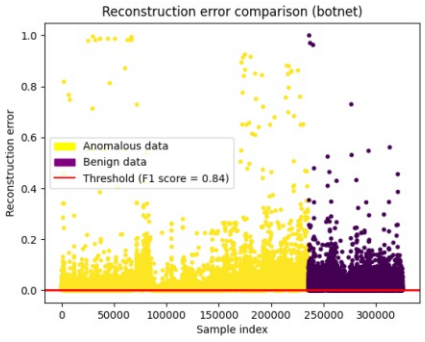

Fig. 2.37: *Reconstruction error and threshold of experiment ISOT_024.*

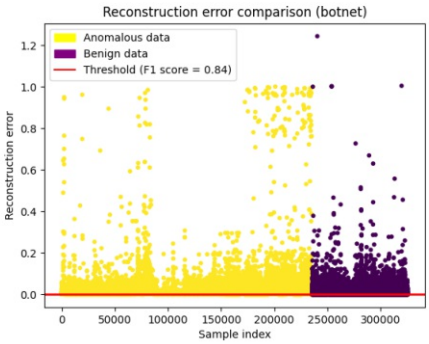

Fig. 2.38: *Reconstruction error and threshold of experiment ISOT_030.*

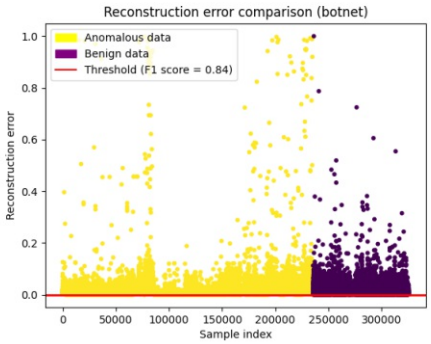

Fig. 2.39: *Reconstruction error and threshold of experiment ISOT_031.*

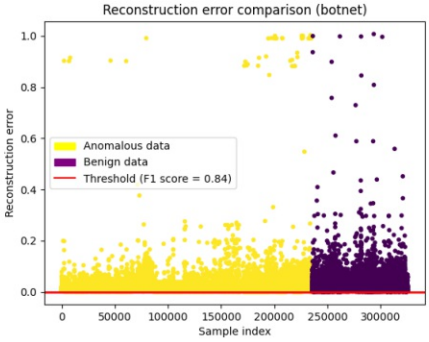

Fig. 2.40: *Reconstruction error and threshold of experiment ISOT_032.*

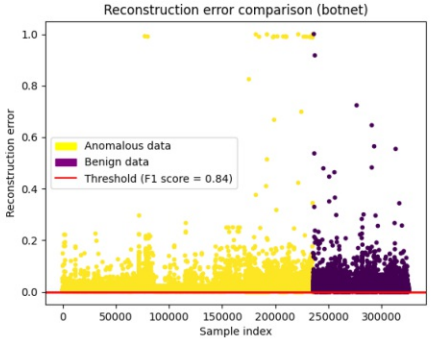

Fig. 2.41: *Reconstruction error and threshold of experiment ISOT_033.*

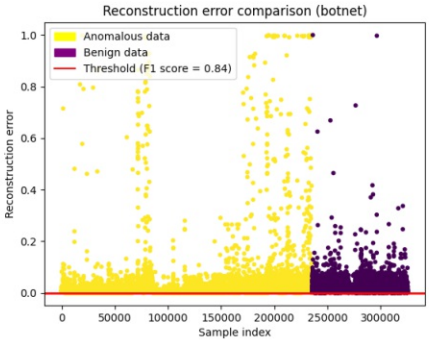

Fig. 2.42: *Reconstruction error and threshold of experiment ISOT_034.*

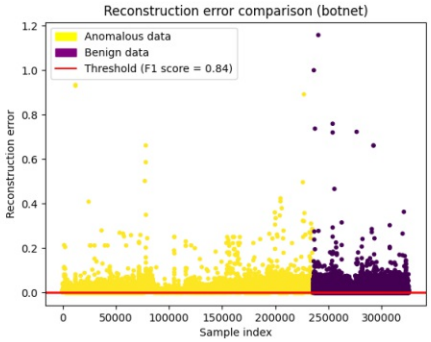

Fig. 2.43: *Reconstruction error and threshold of experiment ISOT_035.*

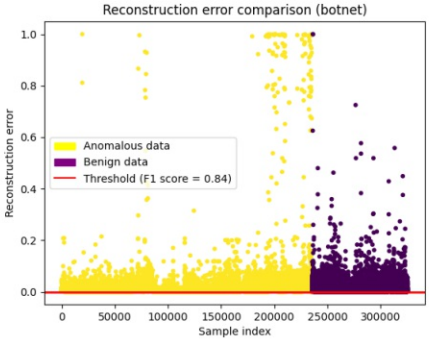

Fig. 2.44: *Reconstruction error and threshold of experiment ISOT_036.*

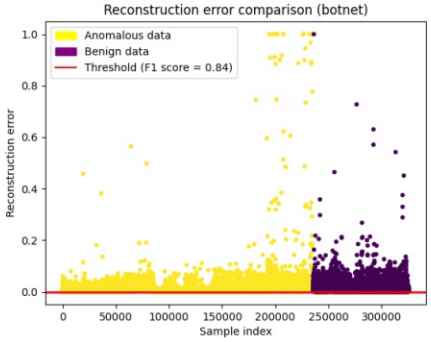

Fig. 2.45: *Reconstruction error and threshold of experiment ISOT_037.*

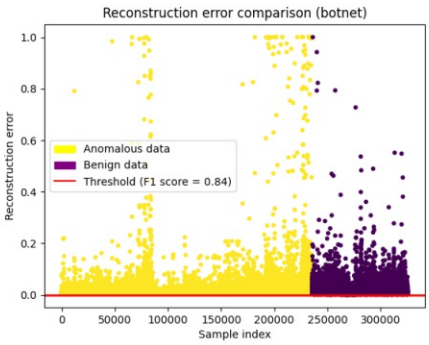

Fig. 2.46: *Reconstruction error and threshold of experiment ISOT_040.*

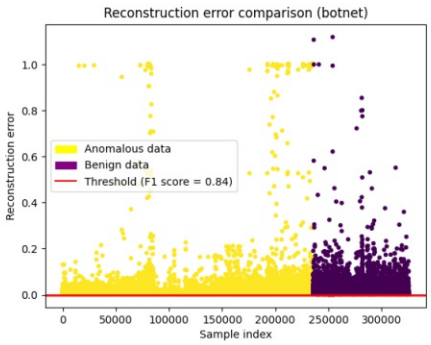

Fig. 2.47: *Reconstruction error and threshold of experiment ISOT_041.*

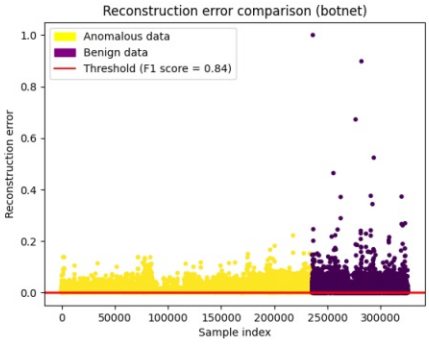

Fig. 2.48: *Reconstruction error and threshold of experiment ISOT_042.*

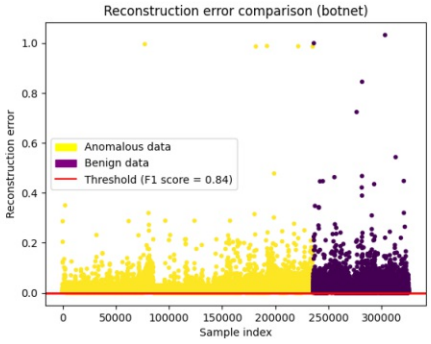

Fig. 2.49: *Reconstruction error and threshold of experiment ISOT_043.*

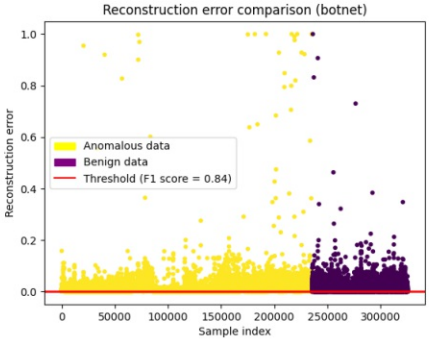

Fig. 2.50: *Reconstruction error and threshold of experiment ISOT_044.*

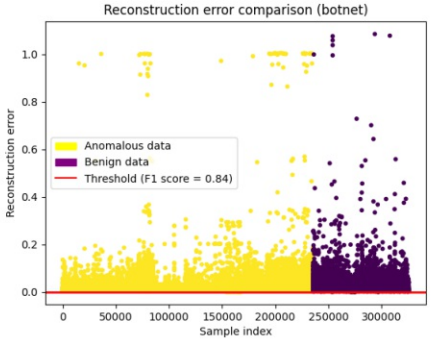

Fig. 2.51: *Reconstruction error and threshold of experiment ISOT_045.*

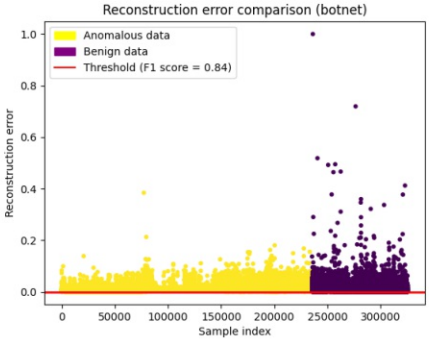

Fig. 2.52: *Reconstruction error and threshold of experiment ISOT_050.*

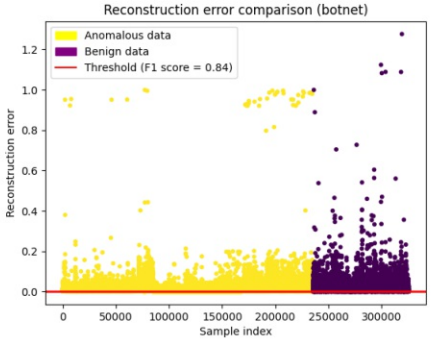

Fig. 2.53: *Reconstruction error and threshold of experiment ISOT_051.*

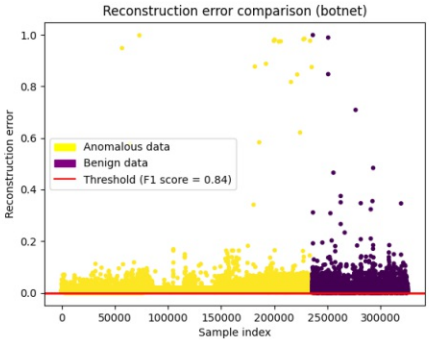

Fig. 2.54: *Reconstruction error and threshold of experiment ISOT_052.*

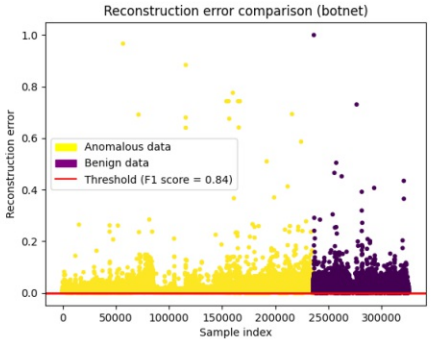

Fig. 2.55: *Reconstruction error and threshold of experiment ISOT_053.*

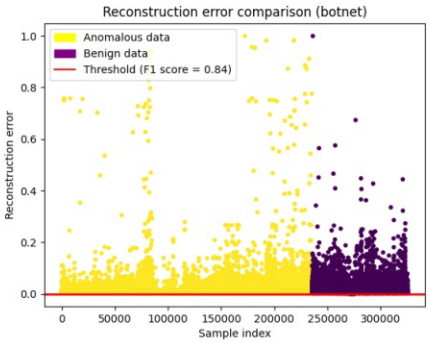

Fig. 2.56: *Reconstruction error and threshold of experiment ISOT_054.*

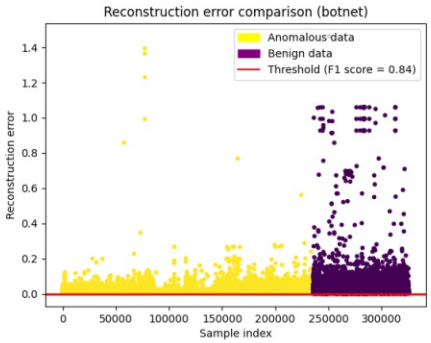

Fig. 2.57: *Reconstruction error and threshold of experiment ISOT_055.*

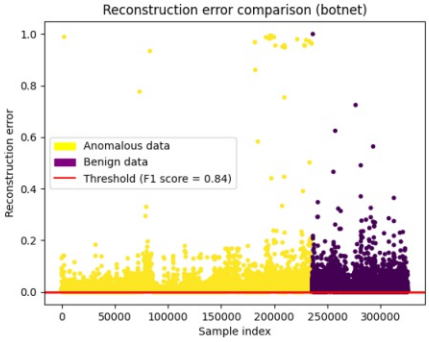

Fig. 2.58: *Reconstruction error and threshold of experiment ISOT_060.*

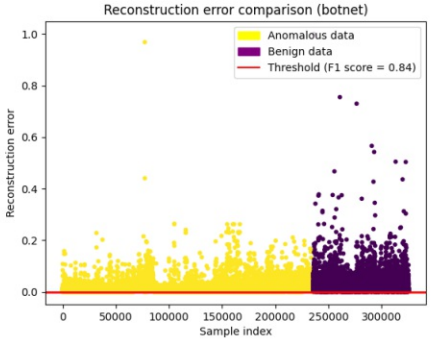

Fig. 2.59: *Reconstruction error and threshold of experiment ISOT_061.*

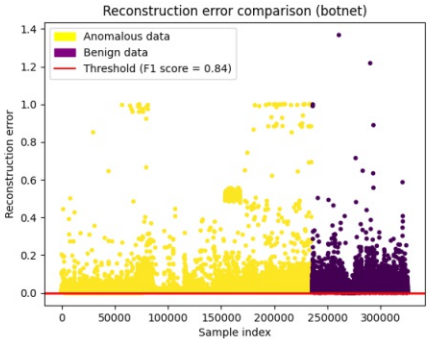

Fig. 2.60: *Reconstruction error and threshold of experiment ISOT_062.*

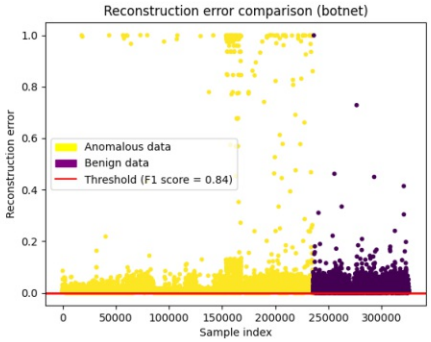

Fig. 2.61: *Reconstruction error and threshold of experiment ISOT_063.*

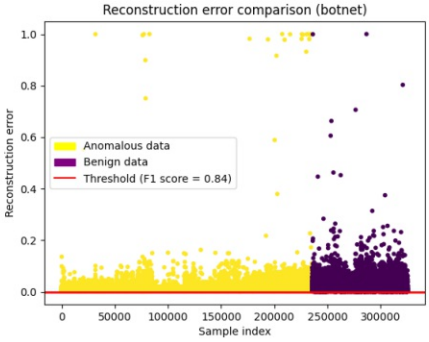

Fig. 2.62: *Reconstruction error and threshold of experiment ISOT_064.*

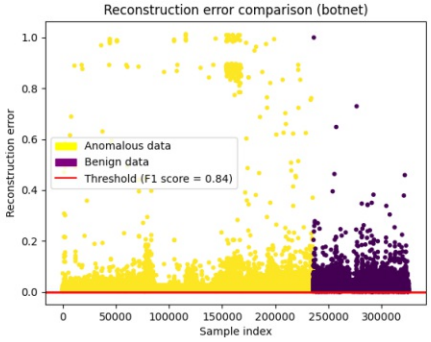

Fig. 2.63: *Reconstruction error and threshold of experiment ISOT_065.*

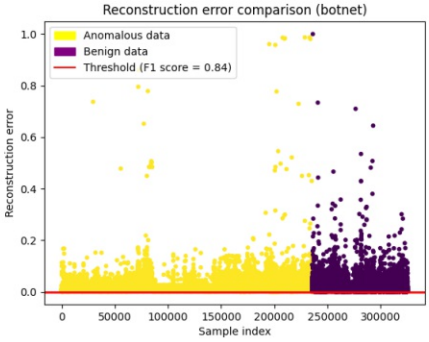

Fig. 2.64: *Reconstruction error and threshold of experiment ISOT_066.*

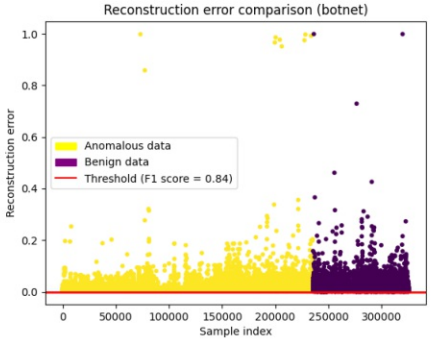

Fig. 2.65: *Reconstruction error and threshold of experiment ISOT_067.*

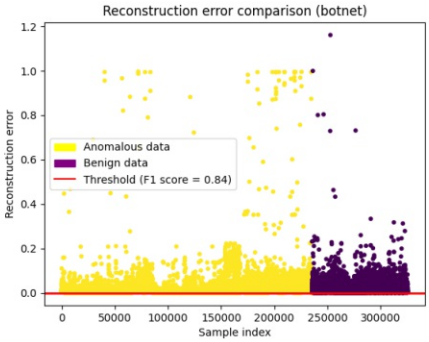

Fig. 2.66: *Reconstruction error and threshold of experiment ISOT_068.*

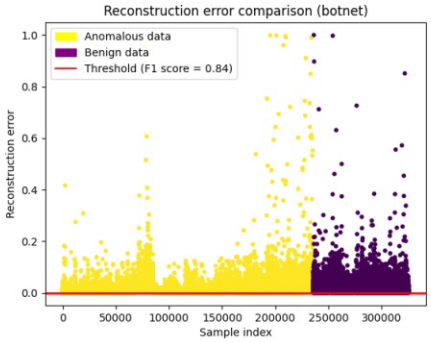

Fig. 2.67: *Reconstruction error and threshold of experiment ISOT_069.*

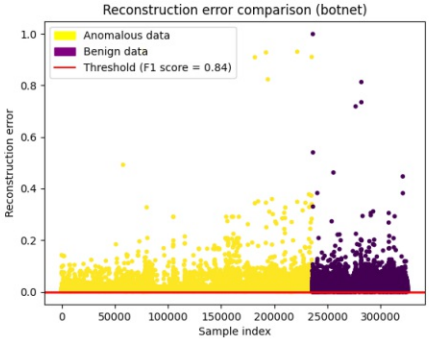

Fig. 2.68: *Reconstruction error and threshold of experiment ISOT_0610.*

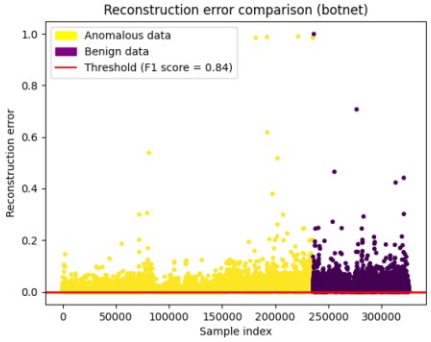

Fig. 2.69: *Reconstruction error and threshold of experiment ISOT_0611.*

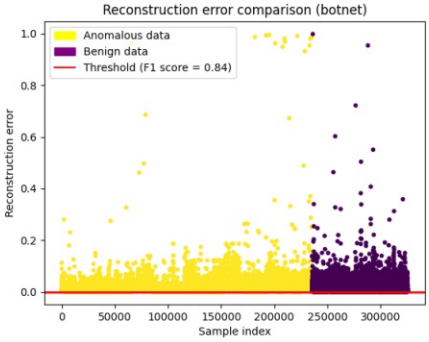

Fig. 2.70: *Reconstruction error and threshold of experiment ISOT_070.*

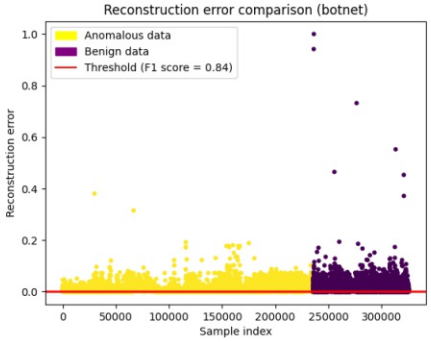

Fig. 2.71: *Reconstruction error and threshold of experiment ISOT_071.*

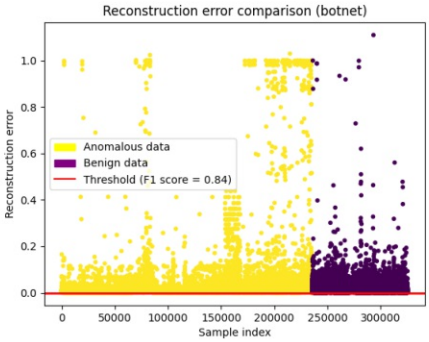

Fig. 2.72: *Reconstruction error and threshold of experiment ISOT_072.*

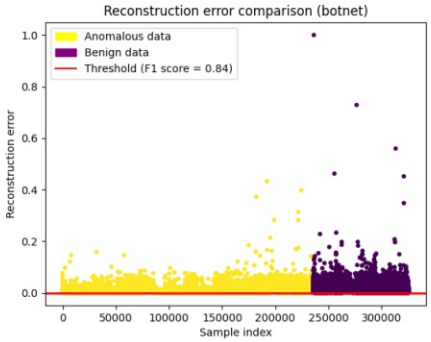

Fig. 2.73: *Reconstruction error and threshold of experiment ISOT_073.*

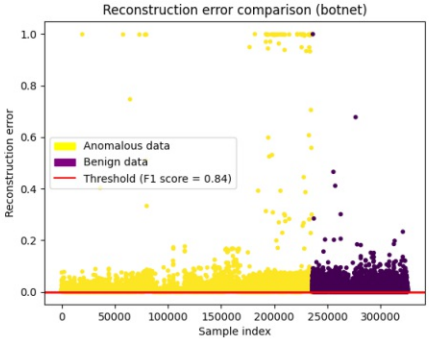

Fig. 2.74: *Reconstruction error and threshold of experiment ISOT_080.*

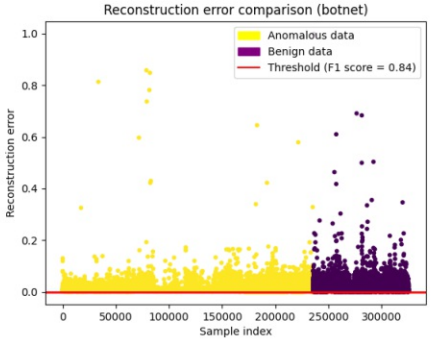

Fig. 2.75: *Reconstruction error and threshold of experiment ISOT_081.*

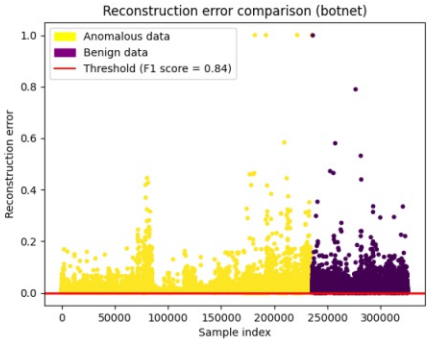

Fig. 2.76: *Reconstruction error and threshold of experiment ISOT_082.*

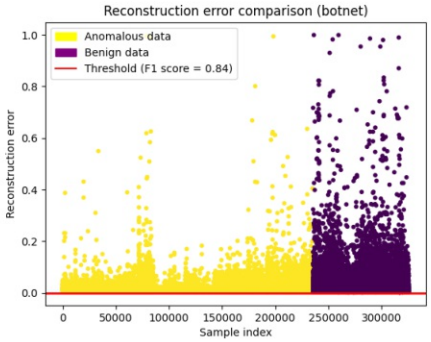

Fig. 2.77: *Reconstruction error and threshold of experiment ISOT_083.*

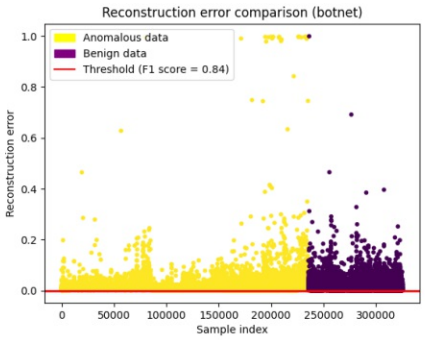

Fig. 2.78: *Reconstruction error and threshold of experiment ISOT_084.*

Bibliography

[1] J. Livingood, N. Mody, and M. O'Reirdan, "Recommendations for the Remediation of Bots in ISP Networks," RFC 6561, Mar. 2012. [Online]. Available: https://rfc-editor.org/rfc/rfc6561.txt

[2] "The real story of stuxnet, how kaspersky lab tracked down the malware that stymied iran's nuclear-fuel enrichment program," https://spectrum.ieee.org/the-real-story-of-stuxnet, Oct 2021.

[3] M. Stevanovic and J. Pedersen, "Machine learning for identifying botnet network traffic," Apr 2013.

[4] "Fbi most wanted," https://www.fbi.gov/wanted/cyber/evgeniy-mikhailovich-bogachev, Oct 2021.

[5] S. Dange and M. Chatterjee, "Iot botnet: The largest threat to the iot network," in *Data Communication and Networks*, L. C. Jain, G. A. Tsihrintzis, V. E. Balas, and D. K. Sharma, Eds. Singapore: Springer Singapore, 2020, pp. 137–157.

[6] J. Margolis, T. T. Oh, S. Jadhav, Y. H. Kim, and J. N. Kim, "An In-Depth Analysis of the Mirai Botnet," in *2017 International Conference on Software Security and Assurance (ICSSA)*, Jul. 2017, pp. 6–12.

[7] Z. Long, L. Tan, C. He, and S. Zhou, "Collecting indicators of compromise from unstructured text of cybersecurity articles using neural-based sequence labelling," *2019 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2019.

[8] S. Abu-Nimeh, D. Nappa, X. Wang, and S. Nair, "A comparison of machine learning techniques for phishing detection," in *Proceedings of the anti-phishing working groups 2nd annual eCrime researchers summit*, 2007, pp. 60–69.

[9] S. Althubiti, W. Nick, J. Mason, X. Yuan, and A. Esterline, "Applying Long Short-Term Memory Recurrent Neural Network for Intrusion Detection," in *SoutheastCon 2018*, Apr. 2018, pp. 1–5, iSSN: 1091-0050.

[10] S. Saad, I. Traore, A. Ghorbani, B. Sayed, D. Zhao, W. Lu, J. Felix, and P. Hakimian, "Detecting p2p botnets through network behavior analysis and machine learning," in *2011 Ninth Annual International Conference on Privacy, Security and Trust*, 2011, pp. 174–180.

[11] B. Abraham, A. Mandya, R. Bapat, F. Alali, D. E. Brown, and M. Veeraraghavan, "A Comparison of Machine Learning Approaches to Detect Botnet Traffic," in *2018 International Joint Conference on Neural Networks (IJCNN)*, Jul. 2018, pp. 1–8.

[12] J. Bhatia, R. Sehgal, and S. Kumar, "Honeynet based botnet detection using command signatures," in *Advances in wireless, mobile networks and applications*. Springer, 2011, pp. 69–78.

[13] C. Li, W. Jiang, and X. Zou, "Botnet: Survey and case study," in *2009 Fourth International Conference on Innovative Computing, Information and Control (ICICIC)*. IEEE, 2009, pp. 1184–1187.

[14] A. Schonewille and D.-J. Van Helmond, "The domain name service as an ids," *Research Project for the Master System-and Network Engineering at the University of Amsterdam*, 2006.

[15] D. Dagon, "Botnet detection and response," in *OARC workshop*, vol. 2005, 2005.

[16] H. Choi, H. Lee, H. Lee, and H. Kim, "Botnet detection by monitoring group activities in dns traffic," in *7th IEEE International Conference on Computer and Information Technology (CIT 2007)*, Oct 2007, pp. 715–720.

[17] J. Goebel and T. Holz, "Rishi: Identify bot contaminated hosts by IRC nickname evaluation," in *First Workshop on Hot Topics in Understanding Botnets (HotBots 07)*. Cambridge, MA: USENIX Association, Apr. 2007. [Online]. Available: https://www.usenix.org/conference/hotbots-07/rishi-identify-bot-contaminated-hosts-irc-nickname-evaluation

[18] T. Strayer, D. Lapsely, R. Walsh, and C. Livadas, *Botnet Detection Based on Network Behavior*, 10 2007, vol. 36, pp. 1–24.

[19] M. Eslahi, R. Salleh, and N. B. Anuar, "Bots and botnets: An overview of characteristics, detection and challenges," in *2012 IEEE International Conference on Control System, Computing and Engineering*, Nov 2012, pp. 349–354.

[20] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, 07 2009.

[21] "What is the mirai botnet?" https://www.cloudflare.com/learning/ddos/glossary/mirai-botnet, Oct 2021.

[22] R. W. Shirey, "Internet Security Glossary, Version 2," RFC 4949, Aug. 2007. [Online]. Available: https://rfc-editor.org/rfc/rfc4949.txt

[23] "Fast flux botnets still wreaking havoc on the internet according to akamai research," https://www.akamai.com/newsroom/press-release/fast-flux-botnets-still-wreaking-havoc-on-internet-according-to-akamai-research, Oct 2021.

[24] "Krebsonsecurity hit with record ddos," https://krebsonsecurity.com/2016/09/krebsonsecurity-hit-with-record-ddos/, Oct 2021.

[25] T. Holz, M. Steiner, F. Dahl, E. W. Biersack, F. C. Freiling *et al.*, "Measurements and mitigation of peer-to-peer-based botnets: A case study on storm worm." *Leet*, vol. 8, no. 1, pp. 1–9, 2008.

[26] J. Leyden, "Ms uses court order to take out waledac botnet," Feb 2010. [Online]. Available: https://www.theregister.com/2010/02/25/ms_waledac_takedown/

[27] "Spam kingpin peter levashov gets time served," https://krebsonsecurity.com/2021/07/spam-kingpin-peter-levashov-gets-time-served/, Oct 2021.

[28] "The life and death of the zeus trojan," https://blog.malwarebytes.com/101/2021/07/the-life-and-death-of-the-zeus-trojan/, Oct 2021.

[29] E. B. Beigi, H. H. Jazi, N. Stakhanova, and A. A. Ghorbani, "Towards effective feature selection in machine learning-based botnet detection approaches," in *2014 IEEE Conference on Communications and Network Security*, Oct. 2014, pp. 247–255.

[30] J. Pickles, T. Stone, and T. Jacques, "Methylation-based algorithms for diagnosis: experience from neuro-oncology," *The Journal of Pathology*, vol. 250, 02 2020.

[31] J. P. M. d. Sá, *Pattern recognition : concepts, methods and applications*. Berlin [u.a.]: Springer, 2001.

[32] T.-T.-H. Le, J. Kim, and H. Kim, "An effective intrusion detection classifier using long short-term memory with gradient descent optimization," in *2017 International Conference on Platform Technology and Service (PlatCon)*, 2017, pp. 1–6.

[33] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ser. ICML'13. JMLR.org, 2013, p. III–1310–III–1318.

[34] S. M. Kasongo and Y. Sun, "A Deep Long Short-Term Memory based classifier for Wireless Intrusion Detection System," *ICT Express*, vol. 6, no. 2, pp. 98–103, Jun. 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2405959519301699

[35] L. Bontemps, V. L. Cao, J. McDermott, and N.-A. Le-Khac, "Collective Anomaly Detection Based on Long Short-Term Memory Recurrent Neural Networks," in *Future Data and Security Engineering*, ser. Lecture Notes in Computer Science, T. K. Dang, R. Wagner, J. Küng, N. Thoai, M. Takizawa, and E. Neuhold, Eds. Cham: Springer International Publishing, 2016, pp. 141–152.

[36] J. An and S. Cho, "Variational autoencoder based anomaly detection using reconstruction probability," 2015.

[37] Y. Bengio, "Learning deep architectures for ai," *Found. Trends Mach. Learn.*, vol. 2, no. 1, p. 1–127, jan 2009. [Online]. Available: https://doi.org/10.1561/2200000006

[38] P. Madani and N. Vlajic, "Robustness of deep autoencoder in intrusion detection under adversarial contamination," in *Proceedings of the 5th Annual Symposium and Bootcamp on Hot Topics in the Science of Security*, ser. HoTSoS '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3190619.3190637

[39] Y. Xia, X. Cao, F. Wen, G. Hua, and J. Sun, "Learning discriminative reconstructions for unsupervised outlier removal," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1511–1519.

[40] S. Garg, A. K. Singh, A. K. Sarje, and S. K. Peddoju, "Behaviour analysis of machine learning algorithms for detecting P2P botnets," in *2013 15th International Conference on Advanced Computing Technologies (ICACT)*, Sep. 2013, pp. 1–4.

[41] M. Stevanovic and J. M. Pedersen, "An efficient flow-based botnet detection using supervised machine learning," in *2014 International Conference on Computing, Networking and Communications (ICNC)*. Honolulu, HI, USA: IEEE, Feb. 2014, pp. 797–801. [Online]. Available: http://ieeexplore.ieee.org/document/6785439/

[42] E. Biglar Beigi, H. Hadian Jazi, N. Stakhanova, and A. A. Ghorbani, "Towards effective feature selection in machine learning-based botnet detection approaches," in *2014 IEEE Conference on Communications and Network Security*, 2014, pp. 247–255.

[43] F. V. Alejandre, N. C. Cortés, and E. A. Anaya, "Feature selection to detect botnets using machine learning algorithms," in *2017 International Conference on Electronics, Communications and Computers (CONIELECOMP)*, Feb. 2017, pp. 1–7.

[44] F. K. Wai, Z. Lilei, W. K. Wai, S. Le, and V. L. L. Thing, "Automated Botnet Traffic Detection via Machine Learning," in *TENCON 2018 - 2018 IEEE Region 10 Conference*, Oct. 2018, pp. 0038–0043.

[45] B. Nechaev, V. Paxson, M. Allman, and A. Gurtov, "On calibrating enterprise switch measurements," in *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 143–155. [Online]. Available: https://doi.org/10.1145/1644893.1644910

[46] S. García, M. Grill, J. Stiborek, and A. Zunino, "An empirical comparison of botnet detection methods," *Computers & Security*, vol. 45, pp. 100–123, 09 2014.

[47] H. Bahşi, S. Nõmm, and F. B. L. Torre, "Dimensionality Reduction for Machine Learning Based IoT Botnet Detection," in *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, Nov. 2018, pp. 1857–1862.

[48] S. Haq and Y. Singh, "Botnet Detection using Machine Learning," in *2018 Fifth International Conference on Parallel, Distributed and Grid Computing (PDGC)*, Dec. 2018, pp. 240–245.

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

[50] J. Kim, J. Kim, H. L. T. Thu, and H. Kim, "Long Short Term Memory Recurrent Neural Network Classifier for Intrusion Detection," in *2016 International Conference on Platform Technology and Service (PlatCon)*, Feb. 2016, pp. 1–5, iSSN: null.

[51] P. Torres, C. Catania, S. Garcia, and C. G. Garino, "An analysis of Recurrent Neural Networks for Botnet detection behavior," in *2016 IEEE Biennial Congress of Argentina (ARGENCON)*, Jun. 2016, pp. 1–6, iSSN: null.

[52] Y. Fu, F. Lou, F. Meng, Z. Tian, H. Zhang, and F. Jiang, "An Intelligent Network Attack Detection Method Based on RNN," in *2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*, Jun. 2018, pp. 483–489.

[53] C. Yin, Y. Zhu, J. Fei, and X. He, "A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks," *IEEE Access*, vol. 5, pp. 21 954–21 961, 2017.

[54] A. H. Mirza and S. Cosan, "Computer network intrusion detection using sequential LSTM Neural Networks autoencoders," in *2018 26th Signal Processing and Communications Applications Conference (SIU)*, May 2018, pp. 1–4, iSSN: null.

[55] C. Torrano-Giménez, A. Pérez-Villegas, and G. Á. Marañón, "An anomaly-based approach for intrusion detection in web traffic," 2010.

[56] S. A. Althubiti, E. M. Jones, and K. Roy, "LSTM for Anomaly-Based Network Intrusion Detection," in *2018 28th International Telecommunication Networks and Applications Conference (ITNAC)*, Nov. 2018, pp. 1–3, iSSN: 2474-1531.

[57] M. Ring, S. Wunderlich, D. Grüdl, D. Landes, and A. Hotho, "Creation of flow-based data sets for intrusion detection," *Journal of Information Warfare*, vol. 16, no. 4, pp. 41–54, 2017. [Online]. Available: https://www.jstor.org/stable/26504117

[58] M. Roopak, G. Y. Tian, and J. Chambers, "Deep Learning Models for Cyber Security in IoT Networks," in *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, Jan. 2019, pp. 0452–0457.

[59] I. Sharafaldin, A. Habibi Lashkari, and A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *ICISSP*, 01 2018, pp. 108–116.

[60] H. Gwon, C. Lee, R. Keum, and H. Choi, "Network Intrusion Detection based on LSTM and Feature Embedding," *arXiv:1911.11552 [cs, stat]*, Nov. 2019, arXiv: 1911.11552. [Online]. Available: http://arxiv.org/abs/1911.11552

[61] B. Min, J. Yoo, S. Kim, D. Shin, and D. Shin, "Network anomaly detection using memory-augmented deep autoencoder," *IEEE Access*, vol. 9, pp. 104 695–104 706, 2021.

[62] N. Moustafa and J. Slay, "Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," in *2015 Military Communications and Information Systems Conference (MilCIS)*, 2015, pp. 1–6.

[63] O. Y. Al-Jarrah, O. Alhussein, P. D. Yoo, S. Muhaidat, K. Taha, and K. Kim, "Data Randomization and Cluster-Based Partitioning for Botnet Intrusion Detection," *IEEE Transactions on Cybernetics*, vol. 46, no. 8, pp. 1796–1806, Aug. 2016.

[64] M. S. Koli and M. K. Chavan, "An advanced method for detection of botnet traffic using intrusion detection system," in *2017 International Conference on Inventive Communication and Computational Technologies (ICICCT)*, Mar. 2017, pp. 481–485.

[65] S. Chen, Y. Chen, and W. Tzeng, "Effective Botnet Detection Through Neural Networks on Convolutional Features," in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, Aug. 2018, pp. 372–378.

[66] B. Rahbarinia, R. Perdisci, A. Lanzi, and K. Li, "Peerrush: Mining for unwanted p2p traffic," *J. Inf. Secur. Appl.*, vol. 19, pp. 194–208, 01 2014.

[67] S. Nõmm and H. Bahşi, "Unsupervised Anomaly Based Botnet Detection in IoT Networks," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Dec. 2018, pp. 1048–1053.

[68] C. Yin, Y. Zhu, S. Liu, J. Fei, and H. Zhang, "An enhancing framework for botnet detection using generative adversarial networks," in *2018 International Conference on Artificial Intelligence and Big Data (ICAIBD)*, May 2018, pp. 228–234, iSSN: null.

[69] G. Szabó, D. Orincsay, S. Malomsoky, and I. Szabó, "On the validation of traffic classification algorithms," in *Passive and Active Network Measurement*, M. Claypool and S. Uhlig, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 72–81.

[70] "French chapter of the honeynet project," https://www.honeynet.org/category/chapters/france, Oct 2019.

[71] G. Vormayr, J. Fabini, and T. Zseby, "Why are my flows different? a tutorial on flow exporters," *IEEE Communications Surveys Tutorials*, vol. 22, no. 3, pp. 2064–2103, 2020.

[72] N. Williams, S. Zander, and G. Armitage, "A preliminary performance comparison of five machine learning algorithms for practical ip traffic flow classification," *Computer Communication Review*, vol. 36, pp. 5–16, 10 2006.

[73] "Transmission Control Protocol," RFC 793, Sep. 1981. [Online]. Available: https://rfc-editor.org/rfc/rfc793.txt

[74] P. Cellier and K. Driessens, *Machine Learning and Knowledge Discovery in Databases: International Workshops of ECML PKDD 2019, Würzburg, Germany, September 16–20, 2019, Proceedings, Part II*, ser. Communications in Computer and Information Science. Springer International Publishing, 2020. [Online]. Available: https://books.google.at/books?id=N4PZDwAAQBAJ

[75] B. Ripley, *Pattern recognition and neural networks*. Cambridge University Press, 1996.

[76] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[77] M. Calzolari, "manuel-calzolari/sklearn-genetic: sklearn-genetic 0.4.0," Apr. 2021. [Online]. Available: https://doi.org/10.5281/zenodo.4661178

[78] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[79] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden,

M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[80] "Github repository," https://github.com/jdb858/autoencoder, Dec 2021.

[81] F. Iglesias Vázquez, A. Hartl, T. Zseby, and A. Zimek, *Are Network Attacks Outliers? A Study of Space Representations and Unsupervised Algorithms*, 03 2020, pp. 159–175.