UNIVERSITY OF TWENTE.

Faculty of Electrical Engineering, Mathematics & Computer Science

Unsupervised Anomaly-Based Network Intrusion Detection Using Auto Encoders for Practical Use

Julik Keijer
j.s.keijer@student.utwente.nl
MSc. Thesis

Supervisors:
DR. J.J. CARDOSO DE SANTANNA
DR. M. POEL
PROF.DR.IR. A. PRAS

University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Contents

Chapter 1

Introduction

Scholars [1] state that the reason security related academic solutions are not being adopted in commercial settings, is that while the results are very promising, they are not representative. The use of these solutions is important, as companies rely more on their digital infrastructure, the need for better protection of digital systems becomes increasingly important. These systems can be protected with Intrusion Detection Systems (IDS). The IDS can monitor hosts (HIDS) and network (NIDS) [2]. Based on the availability of data, this research focuses on network data.

The industry standard for a NIDS is to use signature-based detection mechanisms [3], because it has a low false positive rate and is fast enough for real-time classification [4]. This type of detection is created when an attack has been observed and a signature of the attack can be made. In the other extreme, signature-based detection offers no protection against attacks which have not been seen before, and therefore have no signature yet [5]. A solution to this problem would be to use anomaly-based detection. Anomaly-based detection finds traffic that stands out from normal traffic and therefore does not rely on signatures. The problem with anomaly-based detection is a high False Positive Rate, an unsuitable efficiency for real-time detection and the black box nature of many methods. In addition to that, data is needed to train data-driven anomaly-based methods. Numerous datasets [6]–[12] are presented for the development of anomaly-based methods. These datasets contain labeled data, while in commercial settings it is too expensive to label data.

The methods used for anomaly-based NIDS in academic literature can be divided into two categories: statistical methods [13]–[15] and machine learning methods [16]–[18]. In a preliminary evaluation, machine learning methods show the best results in terms of performance and efficiency. However, the used methods rely on supervised learning, by having a labeled dataset of normal and attack data. Unfortunately, it is costly and currently not viable in a commercial setting to label production data. In unsupervised learning, there is no need for labeled data, since unsupervised methods simply indicate whether certain data points belong together. The state-of-the-art in

unsupervised learning for anomaly-based NIDS focuses on Clustering [19]–[21] and, more recently, on Auto Encoders [22], [23].

The problem this research addresses is the inadequate performance and efficiency of unsupervised anomaly-based detection, to make it suitable for practical use. In commercial use, the IDS signals detections which are then analyzed by a human security specialist. Therefore, it is important that the security specialist is not flooded with false detections, for this the False Positive Rate needs to be reduced. Similarly, it is important that all malicious traffic is detected, for this the True Positive Rate needs to be increased. Finally, the IDS must be able to operate in real-time, for this the time-complexity needs to be reduced. This research aims to improve both the performance, by reducing the False Positive Rate and increasing the True Positive Rate, and efficiency, by reducing the time complexity for unsupervised anomaly-based detection.

From a preliminary evaluation Auto Encoders show results that could make them viable for practical use. A literature study will be conducted on the state-of-the-art of unsupervised anomaly detection to confirm whether Auto Encoders are indeed a viable method for practical use. A method of data processing will be proposed to improve the performance and efficiency of unsupervised anomaly-based NIDS. The selected method from the state-of-the-art will be used for the evaluation of the proposed method. For the evaluation, experiments will be conducted in which the performance and efficiency of the proposed method will be tested on two popular datasets and on data captured from a commercial NIDS.

To structure this research, the following research questions have been formulated:

- RQ 1: Are Auto-Encoders the most viable method for practical use in the state-of-the-art of anomaly-based NIDS?
- RQ 2: What improvements can be made on Auto Encoders for anomaly-based NIDS?
- RQ 3: How can we achieve unsupervised anomaly-based NIDS for practical use?

The scientific contribution of this research is a method for data processing which improves the performance and efficiency of Auto Encoders for unsupervised anomaly-based NIDS. The attention for making anomaly-based detection viable for commercial use came from our partnership with Northwave Nederland B.V. [24].

The remainder of this work is organized as follows. In chapter 2 a literature study is conducted on state-of-the-art in anomaly based NIDS. Chapter 3 evaluates the methods found in the state-of-the-art on a popular dataset, discusses what the requirements are in a practical setting and selects a method viable for practical use. Chapter 4 discusses the methodology used in this research and a method of data preparation to improve the performance and efficiency of unsupervised anomaly detection. Chapter 5 conducts

experiments to demonstrate the improved performance and efficiency of the proposed method on popular datasets. Chapter 6 conducts experiments to demonstrate the improved performance and efficiency of the proposed method on data captured in commercial production. Finally, Chapter 7 gives a recommendation for practical use, discusses the research, draws conclusions and points to future work.

Chapter 2

Finding the state-of-the-art in anomaly based NIDS

In this chapter the answer to research question 1: *Are Auto-Encoders the most viable method for practical use in the state-of-the-art of anomaly-based NIDS?* will be addressed. Background on the material discussed in this research will be given. A survey on the state-of-the-art in anomaly based NIDS will be conducted with a focus on the used data, methods and achieved performance and efficiency. From the survey, the most representative databased and the most viable methods for practical use will be determined and these methods will be selected for further evaluation to confirm the selection.

2.1 Background in Intrusion Detection

In this section, rudimentary background information is given on the topics discussed in this research. The background information serves as a basic introduction into the material and as a definition of the terms used in this work.

2.1.1 Intrusion Detection Systems (IDS)

IDS are deployed to detect attacks, the detection can then be used to mitigate the treat. An IDS consists typically of a number of sensors, that gather information on a system, and a decision engine that analyses the information from the sensors to signal intrusions [2]. The IDS can be divided in two systems, as described below.

Host IDS (HIDS)

A HIDS monitors everything on a host by placing a sensor on every host in the network. The sensor then collects logs of the traffic for that host, the system, running processes, deletion or modification of files and configuration changes [2]. The logs are analyzed in a central server, which can then raise alarms.

Network IDS (NIDS)

A NIDS detects attacks in network traffic by placing sensors such that all traffic passes through it. The sensor then collects traffic packet or flow information from which intrusions can be found. This can, for example, be done using information from the TCP/IP stack, such as IP addresses or packet length.

2.1.2 Types of Detection in NIDS

To detect attacks, a NIDS employs different, or a combination of, methods. Which fall in two categories, signature and anomaly based.

Signature

In signature based detection, attacks are found when the traffic matches the signature of an attack. This is done by giving the IDS a set of rules to match the traffic against. The rules are created by taking attacks and finding discerning features that are not, or less, seen in normal traffic to create the signature.

Anomaly

In anomaly based detection, attacks are found when traffic deviates from what is expected from normal traffic. This is done by comparing new traffic to normal traffic

or attacks from the past. When new traffic is very similar to a previous attack or it is very different from normal traffic it is assumed to be an attack. Therefore attacks that are not exactly the same as previous attacks are still found, because they look more like an attack than normal traffic.

2.1.3 Datasets

To evaluate the used methods, a number of databases have been created. A difference in the used datasets is whether they use real or simulated data. Real data is captured in an environment where actual attacks occur, which are not controlled by the creator of the dataset. This means that there is more noise in the data and the data is often harder to label. Simulated datasets are created by setting up a controlled environment and controlling the launch of attacks. This creates clear attacks, which is necessary for research. However, is less representative of actual network traffic.

2.1.4 Methods for Anomaly based NIDS

To create anomaly-based NIDS, a number of methods can be used, the methods in this survey fall in two categories:

1. Machine Learning methods use models to learn from normal and attack data, to find patterns to be able to classify new traffic;

2. Statistical methods use statistical properties to find traffic that stands out, for example by entropy or Euclidean distance.

Supervised or unsupervised methods

Methods can be supervised or unsupervised. Supervised methods need labeled data. Meaning they need to know whether the data is attack or normal data to be able to create a baseline. With this baseline new inputs can be classified as attack or normal data. As illustrated in figure 2.1 on the left side.

Unsupervised methods do not need labeled data. Meaning, they do not need to know whether the data is attack or normal. Since they can evaluate the data and determine whether new data is more similar to one class or the other. As illustrated in figure 2.1 on the right side.

Machine Learning

Machine learning can be described as follows "A machine learning algorithm is an algorithm that is able to learn from data" [25]. With the rise of Deep Learning, traditional machine learning is now called Shallow Learning, since traditional machine

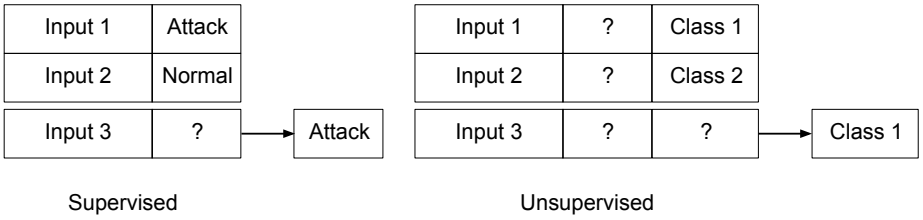


Figure 2.1: Supervised and unsupervised methods

learning consists of one layer, whereas deep learning has multiple. The surveyed Machine Learning methods in this survey are described below.

Decision Trees Decision Trees create a model of the data by finding choices that differentiate the data. It evaluates the characteristics of normal an attack data to build a tree. A simple example is drawn in figure 2.2, where a single choice would determine whether an input is normal or an attack, in reality decision trees are much bigger.

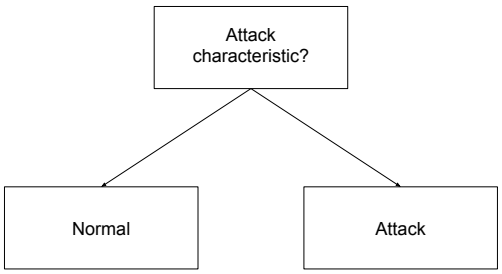


Figure 2.2: Decision Tree

Support Vector Machines A Support Vector Machine finds lines, or support vectors, that differentiate the data. A simple example is drawn in figure 2.3. A SVM can use multiple vectors to differentiate complex groups of data.

Deep Learning

Goodfellow et al. [25] describes Deep Learning as representation learning by introducing representations that are expressed in terms of other, simpler representations. Deep learning enables building complex concepts out of simpler concepts. The methods described below are Deep Learning methods.

Deep Neural Networks A Deep Neural Network has multiple fully connected layers. The layers give an abstraction of the input features by using weights to connect the

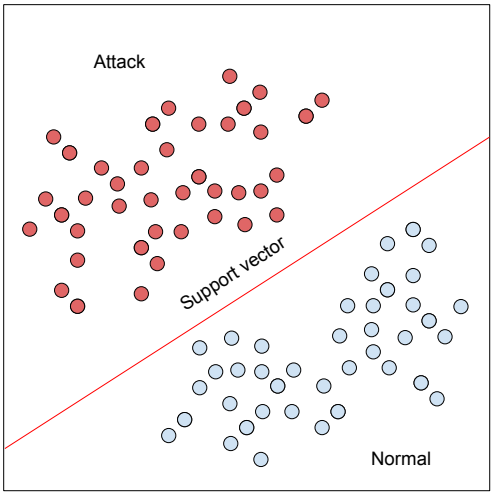


Figure 2.3: Support Vector Machine

hidden layers. The probability of an input being normal or attack data is determined. An example is drawn in figure 2.4. If the output is incorrect, the weights get adjusted, this is called backpropagation. To speed up this process, Extreme Learning Machines (ELM) were created, in ELM the backpropagation is done using matrix inversion, which makes it less accurate, but much faster.

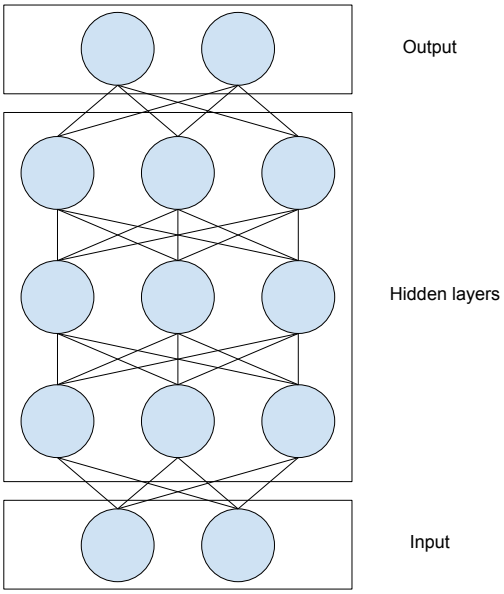


Figure 2.4: Deep Neural Network

Convolutional neural networks A Convolutional Neural Network has multiple convolution and pooling layers. In a convolution layer the matrix of the input data is convolved or combined into a smaller representation. In the pooling the smaller repre-

sentation of the convolutional layer is further reduced by taking the average or maximum value in that part of the matrix. What this means is that a CNN extracts larger patterns from the data. This is useful in image classification, where one pixel does not hold much information, but a group of pixels together form a part of an image.

Recurrent Neural Networks - LSTM In a Recurrent Neural Network each input is not only evaluated on itself, but also the previous inputs. Therefore it is evaluated in the context of the surrounding inputs, which is very useful to detect trends, assuming the surrounding inputs are related. To improve RNN, Long Short Term Memory (LSTM) was created. In LSTM, the model can forget irrelevant data and create a long-term memory for classification, this is very useful when related inputs are not next to each other. An example is drawn in figure 2.5. Here it can be seen that an input is evaluated by the LSTM and the outcome is passed to the next LTSM node to evaluate the next input.

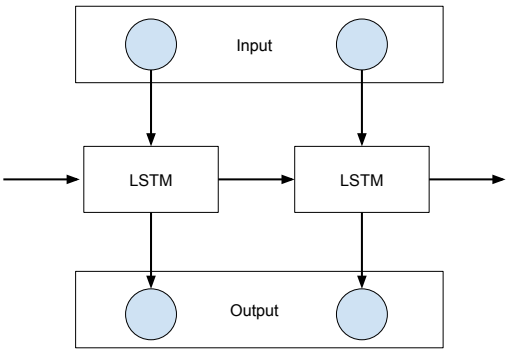


Figure 2.5: Recurrent Neural Network

Deep Belief Networks A Deep Belief Network consists of multiple layers that are trained individually. Just as in Shallow Learning. The layers are trained using a greedy training algorithm that forms connections between each node of each layer. Due to the unsupervised layer, a DBN is suitable for feature selection, as in [26].

Auto Encoders An Auto Encoder consists of two parts, the encoder and the decoder. The encoder creates a representation of the input data and the decoder tries to recreate the input data from the representation. An example is drawn in figure 2.6. When the decoder is able to recreate the input, it means the representation is able to recognize all input data correctly. Which can then be used to classify new inputs. Since the encoder creates the representation without knowing anything about the input data, an auto encoder can work unsupervised. An Auto Encoder. can be used in feature selection [27] or as a classifier [23].

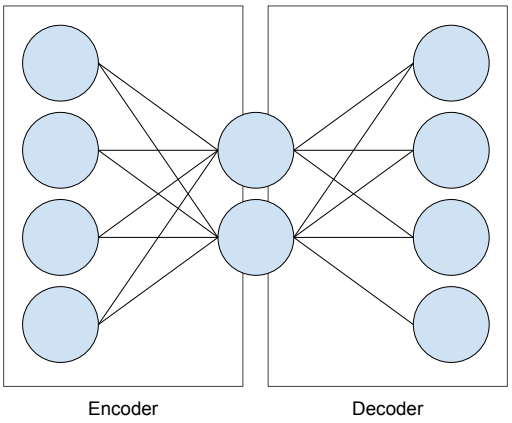


Figure 2.6: Auto Encoder

Statistical Methods

In this section background for the surveyed statistical methods is given.

Clustering In clustering, similar inputs are placed in the same cluster and therefore classify input data as normal or attack. First, a number of clusters are created and then the input data is placed into the cluster where it is closest to. Clustering therefore happens unsupervised.

Sketching / Hashing Sketching is a type of clustering, where hashing of input data is used to create clusters. Hashing means that a mathematical computation is used to transform the input data, for example by making it smaller. In this way, the input does not need to be compared to previous data to determine the cluster, only the hashing is needed to calculate in which cluster the data should be.

Outlier detection In Outlier detection the method finds input points that different in respect to the points around them or in respect to the entire dataset. Popular methods are Local Outlier Factor and Isolation Forest, both used by [28].

Principal Component Analysis Principal Component Analysis tries to describe the dataset using only a few components. For example, a number of the features. When new data cannot be explained by the principal components, it does not resemble normal data and is therefore classified as an anomaly.

2.1.5 Performance metrics

To describe the performance of a method, the following metrics are often used. In table 2.1 a confusion matrix is shown. This table shows how the prediction of a classification

is evaluated against the actual classes of the data.

		Actual	
		Attack	Normal
Predicted	Attack	True Positive (TP)	False Positive (FP)
	Normal	False Negative (FN)	True Negative (TN)

Table 2.1: Confusion Matrix

Accuracy describes the percentage of correctly classified data points with respect to the total amount. This does not give insights into how well each class is classified. Accuracy is calculated as follows:

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

True Positive Rate (TPR) shows the percentage of attack data found. The TPR is also often called Recall or Detection Rate. The TPR is calculated as follows:

$$TPR = \frac{TP}{TP + FN}$$

False Positive Rate (FPR) describes the percentage of normal data in the data predicted as an attack. The FPR is often called the False Alarm Rate. The FPR is calculated as follows:

$$FPR = \frac{FP}{FP + TN}$$

Precision describes the percentage of attack data in the data predicted as attack. It can be calculated by:

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

F1 score describes the harmonic mean of the Precision and the TPR. The F1 score is calculated as follows:

$$F1 = 2 * \frac{Precision * TPR}{Precision + TPR}$$

2.1.6 Conclusions

In this chapter a basic understanding of Signature and Anomaly based NIDS, datasets, 12 anomaly based methods and 5 performance metrics have been given. This background information can be used in understanding the concepts which are discussed in the remainder of this work.

2.2 Methodology in finding the state-of-the-art

This section explains the methodology in finding the literature to conduct a survey and answer the first research question to provide an overview of the state-of-the-art. This section explains the used search engine, used search queries and selection criteria for finding academic literature.

Search Engine

To conduct the survey, Google Scholar [29] has been used. This search engine has been chosen, since it provides a wide range of works. Martin et al. [30] found that Google Scholar holds almost all works of other popular search engines, Scopus and Web of Science. The pitfall of using Google Scholar is that not all works have been peer-reviewed and are of substantial quality.

Search queries

To find relevant works, a number of search queries have been used to get a broad overview of the subject matter. The queries can be found in table 2.2. Where spaces can be read as [OR] and spaces between quotes can be read as [AND]. Therefore, the query: *machine learning "intrusion detection systems"* can be read as *machine OR learning OR (intrusion AND detection AND systems)*.

For each query, the top 20 hits have been scanned. For search queries, broad terms such as *"anomaly network intrusion detection systems"* have been used to get the top cited works and a good overview. Based on prior knowledge and knowledge gained during the creation of the survey, more specific search queries, such as *"intrusion detection locality sensitive hashing"*, have been used to survey works that are less cited, but still relevant to the state-of-the-art.

From these queries an overview of works could be collected. Where the broad search queries returned the largest amount of useful works and surveys, especially when looking at recent years. More specific queries often result in less useful works, however they give the best overview of that specific part of the state-of-the-art. When going from broad to specific queries, there is also some overlap. Queries with ID 2 and 3, from table 2.2, have a number of overlapping works as much of the recent anomaly based research is into machine learning.

Selection Criteria

To collect works for the survey, some selection criteria were used. Since not all works in Google Scholar are of the desired quality, works have been selected based on the number of citations and, if possible on a reputable journal.

ID	Search Query	Hits	Scanned papers
1	signature network intrusion detection systems	167.000	20
2	anomaly network intrusion detection systems	162.000	20
3	machine learning network intrusion detection systems	149.000	20
4	network anomaly outlier	69.000	20
5	machine learning "intrusion detection systems"	54.100	20
6	network intrusion detection clustering	35.400	20
7	unsupervised anomaly based intrusion detection	27.600	20
8	network anomaly detection hashing	24.400	20
9	intrusion detection locality sensitive hashing	17.400	20

Table 2.2: Search keywords

The second selection criterion has been how recent a work is. In the last years the research field in anomaly-based NIDS has advanced significantly. Especially in the field of using machine learning and deep learning for anomaly detection, works from 2019 and 2020 show much more promising results than earlier works. Furthermore, in the search for works a preference has been given to surveys. In these works, the knowledge of the state-of-the-art has been condensed and an overview of the selected works is given, from which the most relevant works for this research have been selected. The consulted surveys are discussed in section 2.3.1.

Third selection criterion is based on the found IDS datasets, discussed in section 2.3.2. Most dataset works ask to be cited when that dataset is used in a research. Therefore the works that cite the dataset research give a good overview of recent and relevant works. Google Scholar gives the option to search for works that cite a certain research. This option was then used to find relevant works for our research. Lastly, for each research the abstract and conclusion is read to determine its relevance. If deemed relevant, the conclusion and results sections are used to fill table 2.5.

2.2.1 Concluding Remarks

In the surveyed works, most of the recent research is into the use of machine and deep learning for anomaly detection. Whereas most the statistical or clustering based methods have been researched earlier, as seen in figure 2.7. As recent as possible works have been selected, to give the best comparison between methods.

All selected works are researches on anomaly-based NIDS, however some works also include research on signature-based NIDS. From all scanned works, 33 academic works have been selected in our survey on the state-of-the-art in Anomaly Based NIDS. The results of the survey will be discussed in chapter 2.3.

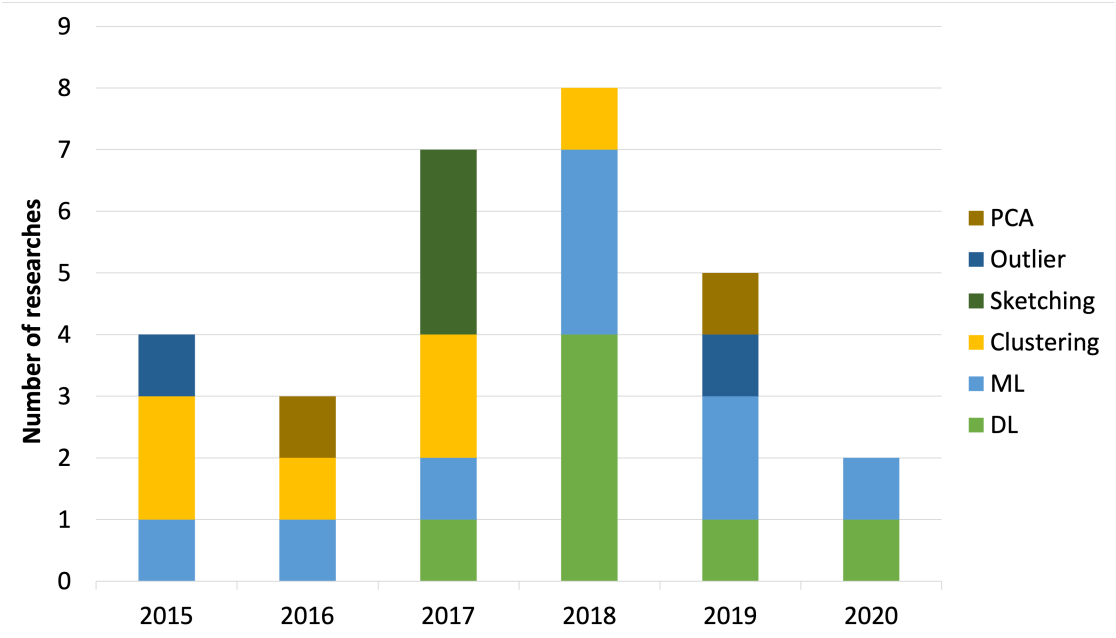


Figure 2.7: Academic works on Method per Year

2.3 State-of-the-art in Anomaly Based NIDS

To answer the first research question and provide an overview of the state-of-the-art, in this chapter, the results of our survey will be discussed. In section 2.3.1 the found surveys are discussed. In section 2.3.2 the IDS datasets used in the surveyed researches are discussed. In section 2.3.3 the methods used in the surveyed researches are discussed, as well as the achieved performance and efficiency per method.

2.3.1 Surveys

A number of surveys have been consulted to answer RQ 1. The works in these surveys have been analyzed and the most relevant works to the broader research have been selected and added to the collection in table 2.5.

Ring et al. [31] show an overview of IDS datasets. The datasets are categorized by size and usage of real or emulated data. For example, Sperotto et al. [7] use a honeypot to capture real attacks. Aldweesh et al. [1] give an overview of the literature in deep learning for IDS. They show the distribution of research into deep learning methods for IDS over the years 2014 up to 2018. They also show the distribution of datasets used in those researches and show that the majority is conducted using the KDD99 dataset or the subsequent NSL-KDD dataset. Terzi et al. [32] give an overview of the usage of big data for real-time anomaly-based NIDS and propose their own approach. Liu et al. [33] give an overview of different machine learning and deep learning methods for anomaly-based HIDS and NIDS. The reviewed works are categorized by the type of data input they use, for example flow or packet inspection.

In table 2.3 the related surveys are summarized and their characteristics are noted. Here it can be seen that Aldweesh et al. [1] and Liu et al. [33] give a detailed taxonomy of Deep Learning and Machine Learning for anomaly-based IDS. While a detailed taxonomy is not given in this document, this survey gives a background of the used concepts in chapter 2.1.

ID	Survey	Main topic	Covered years	Academic works	Taxonomy
1	Ring et al. [31]	IDS Datasets	1998 - 2017	33	No
2	Aldweesh et al. [1]	Deep Learning	2014 - 2018	35	Yes
3	Terzi et al. [32]	Big Data	2014 - 2016	15	No
4	Liu et al. [33]	Statistics and ML	2015 - 2019	26	Yes
5	This survey	Statistics and ML	2012 - 2020	33	No

Table 2.3: IDS survey comparison

2.3.2 Datasets

A selection of the most used IDS datasets was made to show the distribution of IDS research on those datasets. This selection can be found in table 2.4. For each dataset, the research that was at the basis of the dataset is cited and for those researches the number of citations at the time of writing is shown. These citations are not all related to anomaly-based IDS research, however, it estimates the popularity of the dataset. Here it can be seen that the largest part of the researches is done on the NSL-KDD dataset. Interesting to note is that while the CICIDS-2017 dataset is published in 2017, since then there have been a significant number of researches on the dataset.

ID	Dataset	Works in this survey using the dataset	Size	Type of Data	Citations as of 5-11-2020	Publication Year	Citations per year
1	NSL-KDD [6]	[34], [27], [35], [26], [4]	150k points	Simulated	2485	2009	204
2	Twente [7]	[36]	14m flows	Real	152	2009	14
3	Kyoto 2006+ [8]	[37], [22]	93m points	Real	201	2011	18
4	CTU-13 [9]	[32]	81m flows	Simulated	395	2014	66
5	UNSW-NB15 [10]	[38], [39], [35]	2m points	Simulated	603	2015	120
6	CICIDS2017 [12]	[18], [40], [26]	3.1m flows	Simulated	565	2017	188
7	CIDDS-001 [11]	[39]	32m flows	Simulated	65	2017	22
8	Bot-IoT [41]	[42]	72m records	Simulated	96	2018	48

Table 2.4: Dataset researches with number of citations

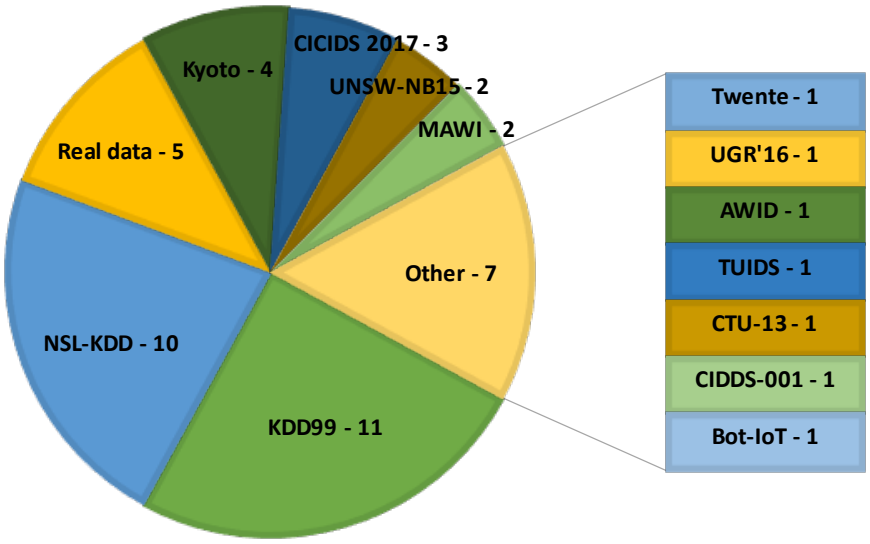


Figure 2.8: Datasets used in the surveyed works

The most used datasets in the surveyed researches are the KDD99 and the subsequent NSL-KDD. The NSL-KDD is based on the KDD99 dataset and was designed to overcome the problems of the KDD99 datasets [6]. The KDD99 dataset contains a lot of duplicate and redundant records and in the NSL-KDD dataset these are left out. However, this means that the NSL-KDD dataset is much smaller. To improve

upon the NSL-KDD dataset a number of datasets were created with simulated traffic. The CICIDS 2017 [12], CIDDS-001, CTU-13, AWID, TUIDS, MAWI lab and Bot-IoT datasets are all simulated datasets with different intentions, for example focusing on botnets, DDoS or IoT traffic.

To evaluate the methods on a more representative dataset, a number of researches work with real data, collected by themselves. This makes it difficult to compare between those researches, since they each use their own dataset. To alleviate this problem, datasets with real traffic captures have been created. The Kyoto 2006+ [8] and Twente [7] datasets were created by collecting honeypot traffic. The Kyoto 2006+ dataset also contains normal traces, whereas the Twente dataset only contains attack traffic. To give researchers the opportunity to evaluate their methods over a longer period of time, the UGR'16 dataset was created by capturing ISP traffic over the period of 4 months. This dataset has both normal and attack traffic. Both the simulated and the captured datasets are labeled to be able to evaluate the methods presented in the surveyed researches.

2.3.3 Overview of methods

In this section, the methods presented in the surveyed works, summarized in table 2.5, will be discussed, as well as their performance and efficiency. To compare the works and evaluate the proposed methods, the works will be discussed per method used for classification of the data.

Decision Trees (DT)

Kim et al. [43] show a hybrid system which is able to classify with low FPR, just like the other DT works. DT are good at classifying large volumes of traffic, for example a DDoS attack. However, since a DT favors a classification to the majority class, there is a problem when trying to find anomalies. Meaning that attacks with a small amount of network traffic, that might be even more detrimental to a network, go undetected. Therefore, the accuracy metrics used to describe the performance does not provide the entire overview. Better already is the TPR, as show by [44]. Here the score of normal packets is not counted, only the malicious traffic score counts and gives a better overview. However, the most important metric is to determine the detection rates per class. This makes it possible to determine whether attacks with a small amount of traffic are also detected.

Zhang et al. [27] display the average of all F1 scores per class. Using Stacked Sparse Auto Encoders (SSAE) for feature selection and eXtreme Gradient Boosting (XGB) for classification, they show that while most classes have a F1 score of over 0.98, the class with the least traffic has a F1 score of 0.7789, resulting in an average F1 score

of 0.9197 over all five classes. Indication that they were able to detect attacks with a small amount of network traffic.

A strong point for using DT, is the efficiency. Verma et al. [39] show the fast execution of Classification and Regression Trees (CART) and XGB, by classifying the CICIDS2017 dataset in 3.2 seconds. Rathore et al. [44], use Hadoop for distributed classification and are able to classify KDD99 in under 1 second. Meaning that these methods can be used for real-time classification.

DT show strong performance but are sometimes lacking in detecting the minority class, however, Zhang et al. [27] show strong performance in both. Strong efficiency is shown by Verma et al. [39].

Support Vector Machines (SVM)

A number of surveyed works, [44], [34], [45], [18] show that the performance and especially the efficiency of SVM is lacking in comparison with other methods like DT.

To overcome this SVM is used in combination with other methods. Kim et al. [43] proposes to first decompose the entire dataset into smaller subsets using a C4.5 DT and then classify the decomposed subsets using One Class SVM (OCSVM) as anomaly detection in the decomposed subsets. Since the subsets will require far fewer support vectors, the testing will go faster and improve efficiency. Because of the outlier detection using the OCSVM, the system is able to find anomalies better than the DT and therefore improve performance for unknown attacks and attacks with a small amount of network traffic.

Khraisat et al. [42] combine the results of C5 DT and OCSVM, using stacked ensemble learning, to gain a higher accuracy 0.9997 than either of the separate methods (C5 DT 0.9330 OCSVM 0.9250). They show this on the Bot-Iot dataset and are able to correctly classify the minority class with a TPR of 1.

Aminanto et al. [46] use SSAE in combination with SVM to achieve a good performance, with high accuracy and low FPR. However, the efficiency is quite low, since it takes 12000 seconds to classify the AWID dataset. The AWID dataset is an IoT dataset with 1 hour of network capture.

Marir et al [26] use a Deep Belief Network (DBN) in combination with a multi-layer SVM to reach high efficiency and classify the CICIDS2017 dataset in 0.1 seconds with a high F1 score.

Shah et al [4] show the performance improvement SVM can have on signature-based detection, when used with the firefly algorithm for feature selection. It can drastically reduce false positives.

SVM show good performance, but poor efficiency. However, Marir et al. [26] show that when SVM is used with another method, the efficiency can be strongly improved.

Deep Neural Networks (DNN)

Vingeswaran et al. [45] and Vinayakumar et al. [18] both show the performance of a DNN with 1 through 5 layers in comparison with shallow machine learning. These researches are evaluated on the KDD99 dataset and show that DNN can improve the classification of normal traffic and therefore have a low FPR. They also perform well on attacks with a lot of traffic, such as probing and DDoS. However, the DNN performs poorly at finding attacks with a small amount of network traffic.

Auto Encoders (AE)

AE can be used as a semi-supervised or even as an unsupervised method. Which means that the method does not require labeled traffic or a normal baseline to learn. As discussed in section 2.1.4. This is valuable for a real world application since it eases the implementation in different networks.

Osada et al. [22] show the already impressive performance of an unsupervised Variational AE (VAE), as well as semi-supervised. Where they label 100 records and show the improved performance on the Kyoto2006+ honeypot dataset. Interestingly the performance on the honeypot dataset with real network captures is better than on the NSL-KDD dataset with simulated data.

Nguyen et al. [23] show the efficiency of VAE for real-time classification of the UGR 16 dataset [47]. A dataset with captures of an entire day over a 4 month period with 110 million records per day. The research uses the following setup: A single graphical processing unit (GPU) GTX 1080Ti with Intel Xeon CPU E5- 2683 (16 cores, 2.1GHz), and 256Gb of RAM. This is hardware is not readily available in most networks. However, with this setup, real-time unsupervised anomaly detection is possible and shown to outperform DT.

SSAE are also used for feature selection in works [27] and [46]. Since SSAE are able to find connections that are not apparent, they are able to find better features for each class and improve the performance.

AE show performance that is comparable to DT when used unsupervised and with good efficiency. Additionally, when used together with other methods AE can strongly improve the performance of those methods.

Convolutional Neural Networks (CNN)

Potluri et al. [35] use CNN for intrusion detection. While the performance for normal traffic was good, the CNN was unable to find the attacks with a small amount of network traffic and therefore the average F1 score over all classes is very low, with 0.4335. This is because the CNN looks for patterns in the data and when those attacks

have a small amount of data, it is unable to find patterns. This makes that CNN is unsuitable for anomaly detection.

Recurrent Neural Networks (RNN)

Anani et al. [48] use RNN - LSTM to classify the KDD99 dataset. The RNN is able to classify with a strong accuracy and low FPR, by learning on trends around the data and, due to the LSTM, by remembering anomalous packets from the past. However, the RNN has a low efficiency and classifies the KDD99 dataset in over 2000 seconds. This makes it unsuitable for real-time classification.

Clustering

Just like AE, as discussed in section 2.3.3, clustering can be used for unsupervised classification. Since clustering groups similar types of traffic together, regardless of classification.

Costa et al. [36], use Optimum-Path Forest Clustering to classify the Twente and NSL-KDD datasets. They show a purity of 0.9577 for the Twente dataset, which means that 95.77% of points were clustered. Since the Twente dataset has no normal traffic, it is only important to group types of malicious traffic together. When introducing normal traffic, with the NSL-KDD dataset a purity of 99.88% is reached.

Other clustering methods do not reach such a high level of accuracy. Researches [49] and [20] reach TPR between 0.90 and 0.93, however, the FPR is between 0.01 and 0.013, which is quite good and better than seen in most of the DT methods as discussed in section 2.3.3.

While Terzi et al. [32] do not focus on the performance of their solution, they show the efficiency of clustering on the large CTU-13 botnet dataset [9]. They use Spark, a distributed computation framework, to speed up their method.

Bhuyan et al. [50] show that by combining tree based clustering with entropy-based outlier detection, very good performance can be reached. An average F1 score of over 0.91 on the KDD99 dataset shows that they are successful at classifying the minority class, however with a higher FPR than other clustering methods.

Al-Jarrah et al. [21] is able to reach a performance of TPR=0.99691 and FPR=0.01049 with only labeling 10 percent of the data. While this requires some labeling, it outperforms all other clustering methods. The method does not outperform on efficiency, however this can be improved using distributed classification.

Clustering shows strong performance, which is comparable to DT, while doing so unsupervised. The performance even improves when a small portion of the data is labeled. The efficiency of Clustering methods is very high, making it suitable for real time anomaly detection.

Sketching

Sketching, or hashing, can be used to improve the efficiency of comparing traffic instances to determine whether they are malicious. Xie et al. [51] use locality sensitive hashing to create buckets with similar traffic. Any new traffic can then quickly be compared by only looking in their bucket. This can be considered as a type of clustering and like clustering it also works unsupervised. Zhang et al. [52] show the performance of different hashing methods, however, as opposed to Xie et al., the efficiency is very poor.

Sketching shows some promising results with performance similar to Clustering, however the efficiency shown in the available academic works is poor and therefore unsuitable for anomaly based detection.

Outlier Detection

Outlier determination can be used to find traffic that deviates from the baseline of normal traffic to find anomalies.

Goldstein et al. [53] propose a very simple method. By creating a histogram for all features and showing traffic that stands out in the histogram, they are able to reach an AUC performance of 0.9999. And are able to classify the KDD99 dataset in 0.06 seconds. This might not be suitable for real data, since malicious and benign traffic is more alike.

Cheng et al. [28] use a combination of isolation forest and Local Outlier Factor (LOF) to find outliers. Like the combination of methods as seen in section 2.3.3 and 2.3.3, isolation forest is used to find global outliers and LOF is used to find local outliers. Since the efficiency of LOF is lower, but the performance is better. The performance of the method of Cheng et al. gets varying results. On the most imbalanced dataset, the accuracy is 0.9999 and the f1 score is 0.7692, which is due to a lower TPR, but a very low FPR. However, on a more balanced dataset the performance is much poorer.

Overall, the performance and efficiency of outlier based methods are not sufficient for anomaly-based detection in the available academic works. While the method of Goldstein et al. [53] works well on the KDD99 dataset, it is doubtful it can be used on real data.

Principal Component Analysis (PCA)

PCA can be used to find important features in datasets. Since network data has a high number of features, it can improve the efficiency of a method to use only the most important features. This might not only improve efficiency, but also performance, as discussed in section 2.3.3 when SSAE are used for feature selection.

Salo et al. [54] use a combination of Information Gain (IG) and PCA for feature selection to improve the performance of their ensemble classifier, based on SVM, k-NN and MLP. They use the Kyoto2006+ dataset to show that the F measure improves from 0.898 without, to 0.984 with IG-PCA and the efficiency improves from 59.02 seconds to 7.07 seconds classification time.

PCA shows a good improvement in performance and efficiency when used together with other methods for classification.

2.3.4 Findings

Decision Trees (DT), Auto Encoders (AE), Deep Belief Networks (DBN), Clustering and Principal Component Analysis (PCA) show good performance with F1 rates of over 0.90, with low False Positive Rates and with efficiency that is sufficient for real-time classification, making them suitable for anomaly-based detection. The other discussed methods are either lacking in performance, in efficiency, or in both.

In the surveyed works, the used datasets vary greatly and therefore the comparison, in many cases, is unclear. In figure 2.9, a number of researches, which presented the F1 score metrics on comparable datasets, are shown. However, in some cases the F1 score over the entire dataset is given and sometimes the F1 score averaged over all classes, like Zhang et al. [27]. When the score over all classes is averaged, the score is lower when attacks with a small amount of network traffic are not found. This is the reason CNN [35] scores relatively low.

In many researches the proposed methods are evaluated against older machine learning methods like Naive Bayes and Random Forest, in those cases the proposed method clearly outperforms. Therefore, methods like Naive Bayes and Random Forest have not been included in the survey.

For most methods, a number of researches is cited to give a good overview of the state of the art for that method. For CNN and RNN only one research is cited. This is due to the fact that for both methods a lacking performance and a clear reason for that lacking performance was found, which makes them unsuitable for anomaly detection. In recent years, more attention has gone to the efficiency of methods and how to make them more efficient. Hybrid methods in the survey perform well on this aspect.

Figure 2.9: F1 scores of different methods on different datasets

ID		Publication	Algorithm	Dataset	Performance	Efficiency	S
1		2014 [43]	Hybrid DT + SVM	KDD99	0.87 TPR, 0.04 FPR	11s KDD99	Yes
2		2015 [16]	J48, RF	Real data	RF 0.966 AUC		Yes
3		2016 [44]	J48, RepTree	KDD99	0.99 TPR, 0.000001 FPR	1s KDD99	Yes
4		2017 [46]	SSAE + SVM	AWID	0.9997 Acc, 0.0001 FPR	12000s AWID	Yes
5	ML	2017 [22]	VAE	NSL-KDD, Kyoto 2006+	Kyoto 0.9533 TPR, 0.0344 FPR; NSL-KDD 0.85991 TPR, 0.12106 FPR		Semi
6		2018 [34]	ELM	NSL-KDD	0.995 Acc, 0.015 FPR		Yes
7		2018 [45]	DNN	KDD99	0.92 Acc, 0.95 F1 score		Yes
8		2018 [4]	SVM + Firefly	NSL-KDD	0.973 Acc, 0.031 FPR		Yes
9		2018 [27]	SSAE + XGB	NSL-KDD	0.9197 avg. F1		No

Table 2.5 continued from the previous page

ID		Publication	Algorithm	Dataset	Performance	Efficiency	S
10		2018 [35]	CNN	NSL-KDD, UNSW-NB15	0.4335 avg. F1		Yes
11		2018 [48]	LSTM	KDD99	0.9885 Acc, 0.009 FPR	2354s KDD99	Yes
12		2018 [26]	DBN + Multi-Layer SVM	NSL-KDD, CICIDS 2017	NSL-KDD 0.9465 F1; CICIDS2017 0.9295 F1	0.1s CICIDS2017	Yes
13		2019 [23]	VAE	UGR'16	0.947 AUC	VAE in real-time	No
14		2019 [40]	DT	CICIDS 2017	0.967 Acc, 0.011 FPR	2.27s CICIDS2017	Yes
15		2019 [42]	C5 DT + OCSVM	Bot-Iot	0.9997 Acc		Yes
16		2020 [39]	CART, XGB	CIDDS-001, UNSW-NB15, NSL-KDD	CART 0.967 Acc, 0.037 FPR; XGB 0.967 Acc, 0.038 FPR	3.2s CICIDS2017	Yes
17		2020 [18]	DNN	CICIDS 2017	0.9 Acc		Yes
18	Clustering	2012 [49]	Sub-space clustering	KDD-99	0.90 TPR, 0.013 FPR	Unclear	No
19		2014 [19]	ASTUTE	MAWI Lab	0.88 F1 score (0.28 improvement)	Real-time	No
20		2015 [36]	Optimum-Path Forest clustering	Twente, NSL-KDD	Twente 0.9577 Purity; NSL-KDD 0.9988 Purity		No
21		2015 [37]	NKICAD	KDD99, Kyoto2006+	0.975 Acc		No
22		2016 [50]	Entropy, Tree based clustering	KDD99, TUIDS	KDD99 0.91 F1; TUIDS 0.97 avg. F1		No
23		2017 [32]	K-NN, Euclidean distance	CTU-13 Botnet traffic	0.96 Acc		No
24		2017 [20]	DEP-SSEC	Real data	0.93 TPR, 0.01 FPR	28 faster than SEC	No
25		2018 [21]	Multi-Layered Clustering	NSL-KDD, Kyoto 2006+	Kyoto 0.99721 TPR, 0.00892 FPR	2.25s kyoto	Semi
26		2017 [51]	LSH	Real data	1.00 Acc	Unclear	No

Table 2.5 continued from the previous page

ID		Publication	Algorithm	Dataset	Performance	Efficiency	S
27	Sketching	2017 [55]	three-dimension reversible sketches	MAWI lab	0.90 TPR, 0.05 FPR		No
28		2017 [52]	L2 LSH	KDD99	0.963 AUC	5000s KDD99	No
29	Outlier	2012 [53]	Histogram-based outlier score (hbos)	KDD99	0.9999 AUC	0.06s KDD99	No
30		2015 [56]	Adaptive Stream Projected Outlier deTector (A-SPOT),	KDD99	0.85 TPR, 0.15 FPR	Linear with data	Yes
31		2019 [28]	Isolation Forest and LOF	Real data	0.96 Acc		No
32	PCA	2016 [15]	PCA - Multivariate Statistical Process Control (MSPC)	Real data	0.96 Acc, 0.05 FPR		Yes
33		2019 [54]	Information Gain + PCA	NSL-KDD, Kyoto 2006+	NSL-KDD 0.981 F1; Kyoto 0.991 F1	2 - 7 seconds	Yes

Table 2.5: Surveyed Works

2.4 Challenges and Directions

From the overview of the state-of-the-art challenges have been identified that will be explored further in the remainder of this research. The four categories that these challenges lie in, are discussed below.

2.4.1 Methods

Several methods in the surveyed works have proven to be viable for real-time anomaly-based NIDS. DTs show good performance and efficiency, however, they fail to find attacks with little traffic. Just as DNN and CNN. One possible solution seems to be the use of DT in combination with SVM. Another is to use SSAE for feature selection before using XGB. PCA also shows good performance and efficiency. The downside of these approaches is the need for labeling, which is very cost expensive in a real world setting.

Unsupervised learning is proposed as a solution in some of the works. With clustering or AE, with good efficiency. However, the performance for most of these works is lacking in respect to supervised methods. The performance increases when semi-supervised learning is introduced, by labeling a small part of the data.

2.4.2 Datasets

A number of different datasets were used to evaluate the proposed solutions in the surveyed works. While newer research tends to use the newer datasets, for example, CICIDS 2017 and UNSW-NB15, the criticized KDD99 and subsequent NSL-KDD dataset are still used very often. Due to varying use of datasets it is difficult to compare the performance of the proposed solutions.

Aside from the comparison problems, the use of some datasets is criticized because these datasets are not representative of real data, therefore some researches use real captured data to ensure that the data is representative. However, these real data captures are not always published and are often only used by one or a handful of researches. Which makes it even harder to compare. Sperotto et al [7] provide a labeled database of real captured traffic from a honeypot as a baseline for real traffic. The limitation here is that the honeypot does not receive normal traffic, therefore the dataset is not suitable to evaluate all methods as a normal baseline is lacking. Song et al. [8] provide a honeypot with labeled data and normal traffic in the Kyoto 2006+ dataset. However, the class balance is not representative. The Kyoto 2006+ dataset contains 53,75 percent normal data and 46.25 percent attack data. In normal traffic, this would more likely be 99.99 percent normal and 0.01 attack traffic. Macia et al. [47] created the UGR'16 data set to evaluate methods over a longer time period.

Using representative data is the biggest challenge in creating a robust anomaly-based NIDS. It will have to be explored whether the simulated or real datasets provide a representative evaluation.

2.4.3 Performance Metrics

As can be seen in table 2.5 the metrics used in the surveyed researches vary a lot. Accuracy (Acc), Area Under Curve (AUC), F1 measure and False Positive Rate (FPR) are among metrics seen in the researches. Aside from the use of different databases, this makes it even more difficult to compare the performance of the surveyed works.

For practical use in a commercial setting, it is important to have a high detection rate, or True Positive Rate (TPR), since all malicious traffic needs to be found. As discussed for decision trees in section 2.3.3, it is important to show the TPR for every classification class. Equally important is a low FPR, since when an alarm is triggered, it takes effort to investigate whether the alarm indicates malicious traffic. When more false alarms are raised, the workload increases and makes it increasingly difficult to investigate all alarms, causing malicious traffic to be found later. Therefore the average F1 score over all classes gives the best indication of how well a method performs, as discussed in section 2.3.3.

2.4.4 Efficiency Metrics

The efficiency of the surveyed results displays how fast a certain solution is in classifying the data. Since the works are evaluated on different datasets and different types of hardware and since some researches simply do not present the efficiency of their solution. It is not possible to give a comprehensive overview. Instead, the presented metrics in the works have been added to table 2.5.

This shows that some solutions have a sufficient classification time to be used in real-time. Other solutions are currently not suitable for real-time classification, as the classification will not be able to keep up with the inflow of traffic. Techniques like distributed classification using Hadoop or Spark, as discussed in section 2.3.3 and 2.3.3, might be able to speed up classification times and make other methods viable for real-time classification. Specialized hardware, as discussed in section 2.3.3, might also decrease classification time significantly.

With differences in datasets, hardware and the use of distributed classification, it is not clear which solutions are suitable for real-time classification. However, it can be seen that methods, such as Decision Trees, clustering, outlier detection and Auto Encoders show low classification times. Hybrid methods can also be used to increase efficiency of methods like SVM.

2.5 Conclusions on the State-Of-The-Art

To answer research question 1: *What is the most viable method for practical use in the state-of-the-art in anomaly-based NIDS?*, an overview of the state-of-the-art in anomaly-based NIDS is given. A survey of 33 researches has been presented with a focus on their used methods, used datasets, performance and efficiency. The overview of the surveyed works show that the best performances are obtained by supervised methods, namely, Decision Trees. To fit the criterion of using unlabeled data, an unsupervised method must be chosen. In this category both clustering and Auto Encoders show good results in terms of performance and efficiency. Auto Encoders showing better performance and Clustering showing better efficiency. The comparison of different methods is complicated by the use of many different intrusion detection datasets, with varying sizes and difficulty of classification. Therefore, the 3 most suitable unsupervised methods: OCSVM, Clustering and Auto Encoders, together with Decision Trees as a benchmark method will be evaluated on the same dataset in the next chapter, chapter 3. From the survey on related academic works, Auto Encoders are shown as the most viable method for practical use in terms of performance, and while the efficiency is lower than clustering, the efficiency is sufficient for practical use.

Chapter 3

Evaluation of the state-of-the-art

In this chapter the datasets and methods found in the state-of-the-art, discussed in chapter 2, will be discussed. The properties and available attacks of the popular UNSW-NB15 and CICIDS 2017 datasets will be discussed. An experiment will be conducted with 4 methods selected from the state-of-the-art to determine which method will be suitable for use in the remainder of this research. Decision Trees, Clustering, Support Vector Machines and Auto Encoders will be evaluated on the UNSW-NB15 dataset for their performance and efficiency. Finally, the requirements for practical use will be discussed.

3.1 Popular IDS datasets

To explore methods for intrusion detection. A number of popular IDS datasets have been used. In the previous chapter, Chapter 2.2. Some of the most used datasets and their corresponding works have been discussed. In this section, a deeper look is taking into these datasets.

3.1.1 UNSW-NB15

The UNSW-NB15 dataset consists of captures spanning two days in which attacks are launched with normal traffic for reference. Table 3.1 shows the distribution of different attacks and normal data. The ratio between normal and attack data is roughly 10:1 and the difference between normal data and the smallest attack category is much larger. Therefore, classification for small classes is a bigger challenge. The attacks are divided into 9 categories to give an overview of relevant attacks and while the worm and backdoor categories are the smallest in number of records, they are also some of the most dangerous categories.

Attack categories	Number of records
Normal	2218764
Generic	215481
Exploits	44525
Fuzzers	24246
DoS	16353
Reconnaissance	13987
Analysis	2677
Backdoor	1795
Shellcode	1511
Backdoors	534
Worms	174

Table 3.1: UNSW-NB15 test data

Figure 3.1 shows the principal component analysis of the UNSW-NB15 dataset. This shows that some exploit and DoS connections are very different from the bulk of normal data and therefore easier to discern. However, the other attack categories are harder to discern from normal data, since their primary component overlap. Therefore these attack categories will be harder to classify.

Figure 3.2 shows the principal component analysis of the training set and figure 3.3 shows the principal component analysis of testing set and of the UNSW-NB15 dataset. The component analyses of both sets are similar and they are both similar to the analysis of the entire dataset, so they are representative of the entire dataset. The only attack category that is easier to separate from the normal data is the generic category.

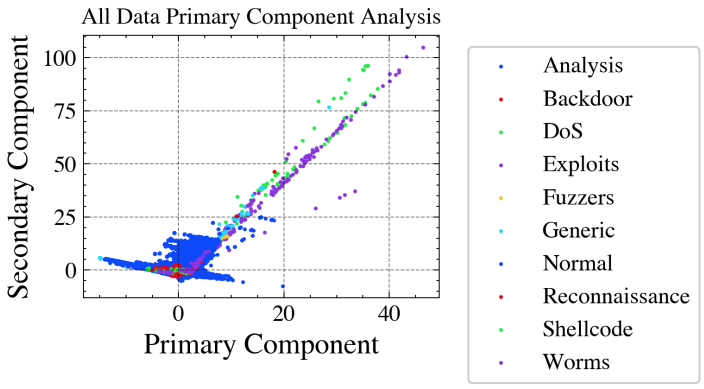


Figure 3.1: UNSW-NB15 Principal Components

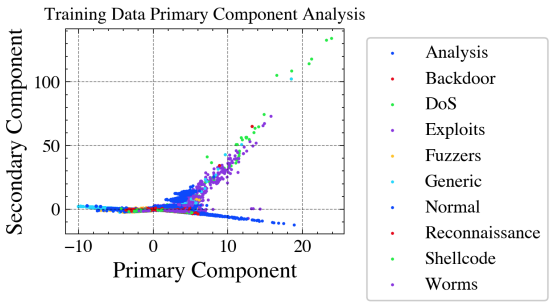


Figure 3.2: UNSW-NB15 Principal Components training set.

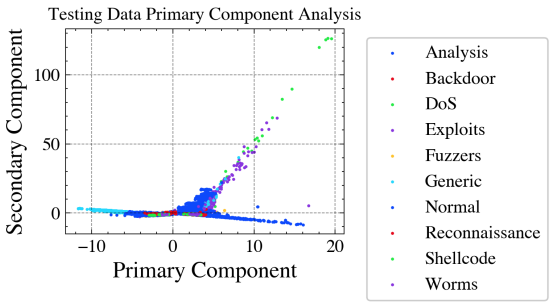


Figure 3.3: UNSW-NB15 Principal Components testing set.

3.1.2 CICIDS-2017

The CICIDS-2017 dataset is a synthetic dataset captured over 5 days. During these 5 days, several attacks have been launched to give a representation of the threat landscape.

In table 3.2 the distribution of records with different types of labels is shown. Here the ratio between normal and attack data can be seen. The biggest difference in proportion is between the labels BENIGN (normal) and Heartbleed (attack) in a ratio of about 160000:1.

Figure 3.4 the Principal Component Analysis is shown. It can be seen that some of the normal data deviates greatly from the rest of the data and the attacks show similarities to the normal data. Making the classification more difficult. Since the normal data also differs greatly, it will be harder to create a normal baseline for the CICIDS 2017 dataset.

Labels	Number of records
BENIGN	1743179
FTP-Patator	7938
SSH-Patator	5897
DoS slowloris	5796
DoS Slowhttptest	5499
DoS Hulk	231073
DoS GoldenEye	10293
Heartbleed	11
Web Attack Brute Force	1507
Web Attack XSS	652
Web Attack Sql Injection	21
Infiltration	36
Bot	1966
DDoS	128027
PortScan	158930

Table 3.2: Number of records per attack.

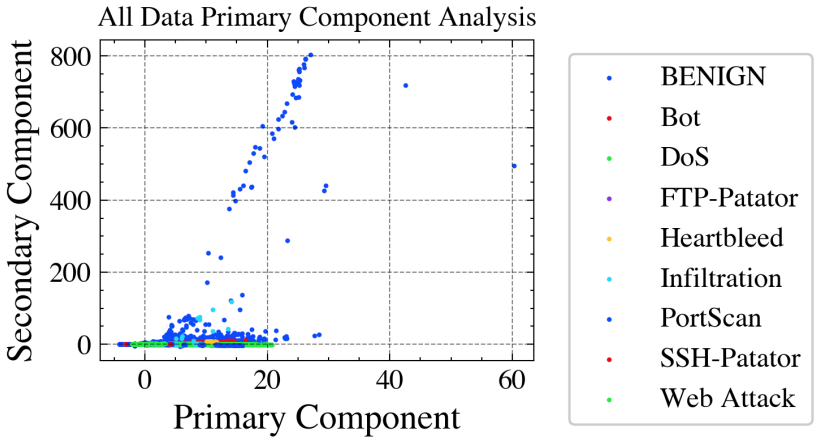


Figure 3.4: CICIDS 2017 Principal Components

3.2 Attack exploration

In this section, the network attacks available in the selected IDS datasets will be explored. This will be done using the MITRE ATT&CK Enterprise framework, which is discussed below. Then the attacks present in the datasets will be evaluated. The framework will be used to evaluate whether the datasets give a comprehensive overview of the threat landscape.

3.2.1 MITRE ATT&CK framework

The MITRE ATT&CK framework is proposed by Strom et al. [57]. They describe the framework as a "curated knowledge base and model for cyber adversary behavior". Meaning that the framework tries to define behavior of attackers and from that categorizes attacks used by those attackers. MITRE ATT&CK defines multiple frameworks, namely for Enterprise networks, Mobile networks and Industrial Control Systems. In this work, the MITRE ATT&CK Enterprise framework will be used to categorize network attacks.

Tactics

The MITRE ATT&Ck enterprise framework divides attacks into 14 categories, called tactics. These tactics define a goal for the attacker and consist of all techniques for acquiring that goal. For example, the tactic reconnaissance has as goal to gather information about the target and consists, among others, of the technique scanning. By scanning a network, possible weak points could be found.

The tactics are divided and set up to give a clear overview of the process of the attacker and their path through a network. First Reconnaissance is needed to gain a better understanding of the network. Followed by access to the system and maintaining that access. Then moving through the system and ultimately acquiring and exfiltrating data and damaging the system.

In table 3.3 The 14 tactics in the MITRE ATT&Ck enterprise framework, their goals and associated types of attacks are noted. For each of the selected IDS datasets, it is displayed which attacks are present. The table illustrates that while a large percentage of tactics are present in both datasets, many tactics are still missing. This means that even using two popular datasets does not give a complete overview of the threat landscape.

The tactics seen most often in IDS datasets are Reconnaissance, Initial Access and Impact. These tactics correspond to attack such as scanning, fuzzing, exploits, shellcode usage and DoS. While it gives a good basis to be able to defend against these attacks gives a good basis, there are many more attacks being executed in the current threat landscape. Attacks from the exfiltration tactic are hard to detect, but the impact of losing sensitive data might be much higher than of a network scan.

Table 3.3: Tactics and their presence in the selected databases.

ID	Tactic	Goal	Types of Attacks	1.[a]	2.[b]
1	Reconnaissance	Find information	Scanning, fuzzing, probing	x	x
2	Resource Development	Not in Network			
3	Initial Access	Achieve foothold	phishing	x	x
4	Execution	Run malicious code	Exploits, shellcode	x	
5	Persistence	Maintain foothold	Create a backdoor	x	x
6	Privilege Escalation	Gain higher permissions			
7	Defense Evasion	Avoid detection			
8	Credential Access	Gather accounts and passwords	Man-in-the-middle, network sniffing		
9	Discovery	Gain understanding of the system	Network service scanning		x
10	Lateral Movement	Move through the system	Service hijacking		
11	Collection	Gather important data	Email collection		
12	Command and Control	Control compromised systems	Receive commands for a central server		x
13	Exfiltration	Extract important data	Data transfer		
14	Impact	Disrupt systems	DoS, Data manipulation	x	x

[a] UNSW-NB15
[b] CICIDS-2017

3.3 Which methods perform

In this section, the methods that showed promising results in section 2.3.3, will be tested on the UNSW-NB15 dataset, discussed in the previous section, section 3.1. This basic evaluation will be conducted to demonstrate the strong and weak points of each method. The methods could be tuned and improved further to obtain better results, however for this evaluation basic settings are used.

All methods are trained and tested on the UNSW-NB15 training and testing set, as discussed in 3.1.1. The training set consists of 175341 rows and 44 features, the testing set consists of 82332 rows and 44 features. For the OCSVM and AE methods, only the normal data is used for training, which are 56000 rows. In the testing set, all data is labeled either as "Normal" or "Attack", to create a binary classification. For all methods, the time needed for training and testing is recorded.

3.3.1 Decision Trees

A Decision Tree learns certain characteristics about the data to make a classification. At every leaf in the tree, new data is weighed to make a decision. For example, on the duration of a connection. Then the model can predict whether a connection is normal or and attack and which type of attack.

Decision Trees are a supervised method and therefore require labeled data. Decision Trees are also sensitive to imbalanced training sets. If there is one class of, for example, normal data which is much larger than the attack classes, the model will be inclined to predict more new connections as normal data. It is therefore important that each class is the same size in the training set. This can be done by decreasing the amount of connections in the largest class or increase the amount in the smallest class. The latter can be done by synthetically creating more connections that lie between the connections in a smaller class and can be done using Synthetic Minority Oversampling Technique (SMOTE) [58].

Action	Total time since start
SMOTE dataset	0:01:23.784247
Training	0:01:25.663109
Testing	0:01:25.682749

Table 3.4: Decision Tree efficiency

		Actual	
		Attack	Normal
Predicted	Attack	35902	3093
	Normal	9430	33907
		TPR = 0.7919	FPR = 0.0836

Table 3.5: Decision Tree confusion matrix on the UNSW-NB15 test set

Table 3.4 shows the time needed for the model to make a classification. The training time is a little less than 2 seconds and the testing time is a little less than 0.02 seconds. The time for Decision Trees is very low, however the time needed to balance the data is much larger and when the testing data would contain more than 2 classes, this time would increase. Table 3.5 shows the classification results and demonstrates that decision trees are able to reach a high TPR and a low FPR with little tuning.

3.3.2 OCSVM

One Class Support Vector Machines (OCSVM) draw vectors between the data and depending on which side of a vector a data point falls determines the classification. This is the supervised version of OCSVM, the method can also be used both in an unsupervised manner. In that case, the method is trained on normal data and new data is scored on how similar it is to the normal data. This can then be used to detect anomalies.

The time complexity of OCSVM is large and therefore it is advised to reduce the number of features with which to train or the number of samples to train on.

Action	Total time since start
Training	0:02:32.367852
Testing	0:04:23.562785

Table 3.6: OCSVM efficiency

		Actual	
		Attack	Normal
Predicted	Attack	39774	21968
	Normal	5558	15032
		TPR = 0.8774	FPR = 0.5937

Table 3.7: OCSVM confusion matrix on the UNSW-NB15 test set

Table 3.6 shows the time needed for training and testing, this shows that with the large number of features, the efficiency of OCSVMs is very low. Not only is the training time of over 2 minutes and 32 seconds, very long, the testing time is also much longer than the other methods. While the TPR is the highest overall methods, the FPR is also very high, indicating that the classification favors the attack class.

3.3.3 Clustering

Clustering groups similar data together based on the distance between the data points, for example the Euclidean distance between two connections can be used to determine their similarity. The clustering method then determines a centroid for each cluster and any new data point is assigned to the cluster of the closest centroid.

Finding the right parameter K, which determines the number of clusters is a known difficult problem. In this evaluation K=2 is chosen. To create a cluster for normal and attack data.

Action	Total time since start
Training	0:00:00.854869
Testing	0:00:00.896214

Table 3.8: K-means Clustering efficiency

		Actual	
		Attack	Normal
Predicted	Attack	30057	10810
	Normal	15275	26190
		TPR = 0.6630	FPR = 0.2922

Table 3.9: K-means clustering confusion matrix on the UNSW-NB15 test set

In table 3.8 the high efficiency of k-means clustering is shown. Both the training and testing time are low. In table 3.9 the classification results are presented which show that the method is better able to classify than OCSVM, but is lacking in respect to Decision Trees.

3.3.4 Auto Encoders

An Auto Encoder tries to reconstruct the input data as best as possible. When a model is trained on a certain input, it will be able to reconstruct that input or data similar to that input. When dissimilar data is used, it will not be recreated as well. The measure in which the model fails to recreate the data is called the reconstruction error. When the model encounters unfamiliar data, the reconstruction error for that data will be higher. This can then be used to find anomalies. When a dataset contains a large amount of normal data and only a small amount of anomalous data. The reconstruction error for the normal data will be low and the reconstruction error for the anomalous data will be higher and therefore be identifiable as an outlier. To calculate the reconstruction error, the Mean Squared Error (MSE) formula is used. This calculates the difference between the input and output data, For the implementation of our Auto Encoder model. Keras [59] has been used. Keras is a python deep learning API, which can be used for fast implementation of deep learning models. Adam was used as an optimization function, as presented by Kingma et al. [60]. ReLu is used as the activation function in the hidden layer and the sigmoid activation function is used in the in and output layers. A summary of used Auto Encoder model is shown in table 3.10.

Layer	Neurons	Activation function
1	64	Sigmoid
2	16	Sigmoid
3	64	Sigmoid

Table 3.10: The used Auto Encoder Model.

Action	Total time since start
Training	0:00:11.546303
Testing	0:00:12.354565

Table 3.11: Auto Encoder efficiency

		Actual	
		Attack	Normal
Predicted	Attack	31781	6915
	Normal	13551	30085
		TPR = 0.7011	FPR = 0.1869

Table 3.12: Auto Encoder confusion matrix on the UNSW-NB15 test set

Table 3.11 shows the efficiency of the Auto Encoder model, this shows that the training and testing times are higher than clustering, but much lower than OCSVM. The classification results, shown in table 3.12, demonstrate that the performance is better than clustering or OCSVM. Therefore Auto Encoders show the best results of the unsupervised methods and are the closest to Decision Trees, which is a supervised method.

3.4 Practical requirements

By conducting the research in collaboration with Northwave [24], many of the important aspects of practical use became clear. Desirable is a 100 percent True Positive Rate, however a low False Positive Rate is essential as well. When the amount of False Positives is too high, not every alert can be investigated and therefore attacks will be missed, regardless of the True Positive Rate. Therefore, it is important to have the least number of False Positives and at least as low that they can be investigated faster than they come in. For example, if analyzing 10 detections takes 1 hour, then that is the limit. Then the highest True Positive Rate is desirable to detect as many attacks as possible.

Furthermore, it is important to quickly report on detections, however it is not necessary to do so instantly. A five-minute window in which network data is classified and attacks are found is sufficiently fast.

3.5 Conclusions on selecting a method

In this chapter an analysis on the datasets and methods surveyed in chapter 2 have been shown. The attacks present in the surveyed popular datasets are presented and section 3.1 demonstrates that the classification of the popular datasets is difficult due to the similarity of the data. Section 3.3 shows a preliminary experiment on the UNSW-NB15 of 4 methods found in chapter 2. This chapter shows that in a comparison of simple implementations of the models, Auto Encoders show results that could make them viable for practical use, in terms of both performance and efficiency. The requirements for practical use have been discussed in **??**. This analysis, together with the survey on the state-of-the-art shows that Auto Encoders are a viable method of practical use in unsupervised anomaly-based NIDS and will be selected to demonstrate the method proposed in the next chapter, chapter 5.

Chapter 4

Methodology

In this chapter, the methodology for conducting this research will be discussed. The methodology for conducting experiments on two popular datasets using Auto Encoders will be discussed. A method for data preparation will be presented, which improves the performance and efficiency of unsupervised anomaly-based detection and the methodology for applying this method will be presented. Finally, the methodology for creating a dataset from data captured in a commercial NIDS, with service-specific features, will be discussed.

4.1 Research Methodology

To conduct the rest of this research and find answers for the posed research questions, this chapter describes the used methodology. First the methodology for conducting the experiments and using Auto Encoders is described.

In the next section, section 4.3 a method for data preparation will be proposed in which network data will be split per service protocol in the application layer. Then an Auto Encoder model will be used per service. Meaning that there will be multiple Auto Encoder models used, which can be executed in parallel. For the training of the Auto Encoder models, the training data is split in a training and validation part, to confirm that the model does not overfit and this validation part can also be used for parameter tuning. The model is then only trained on normal data.

The proposed method of splitting data per service is evaluated on two popular datasets, the UNSW-NB15 and CICIDS 2017 dataset. Using these datasets, the performance and efficiency of the proposed method will be evaluated in terms of True and False Positive Rates and time complexity. The datasets will also be used to find the optimal parameters and configurations of the models.

To further evaluate the proposed method and determine the viability for practical use, a dataset will be created from network data captured in a commercial NIDS. The obtained parameters and configurations from the popular datasets will be used on the real dataset. To evaluate the performance of the proposed method on the real dataset, attack and benign data from the UNSW-NB15 dataset will be inserted in the captured network logs.

4.2 Methodology for using Auto Encoders

In this section, the methodology for using Auto Encoders in the remainder of this research will be explained. The background on using Auto Encoders for anomaly detection can be found in section 2.1.

4.2.1 Data preparation

Before use, the data is transformed. Categorical features are one-hot encoded and continuous features are transformed to have a Gaussian distribution. Peterson et al. [61] show the advantages of Gaussian distributed preprocessing of data using Ordered Quantile normalization. For this Scikit-learn's QuantileTransformer [62] has been used. Transforming to a Gaussian distribution gives more robustness to extreme outliers and creates more difference between data points which are close to the median. This makes it easier to discern between data points that are close together but with some differences.

For all used datasets label features are excluded from training and the model is only trained on normal data. This includes actual labels, such as "attack" or "normal" or "DoS", but also features that can directly be correlated to one of these features, for example, IP addresses, source port numbers, timestamps or connection IDs.

The training data is split in a training and validation set, where the validation set is 0.2 of the total training data. The validation set is used in training of the model to make sure the model does not over-fit and to be able to tune the hyper-parameters.

4.2.2 Methodology

The used Auto Encoder is a Deep Sparse Auto Encoder. For the implementation of this Auto Encoder model Keras [59] has been used. Keras is a python deep learning API, which can be used for fast implementation of deep learning models. Adam was used as a back-propagation function, as presented by Kingma et al. [60].

The formula used as the loss function is Mean Squared Error and to prevent over-fitting and create sparsity in the hidden layers, L1 regularization is used. In the in- and output layers, no regularization is used. The exponential linear unit (ELU) is used as the activation function in the hidden layers. The sigmoid activation function is used in the in and output layers. A summary of the entire model is then given in table 4.1.

Hyper parameter tuning

To find the optimal parameters for the model, a grid search is performed. In the grid search the α parameter of Adam is chosen between [0.5e-4, 1e-4, 2e-4, 5e-4] and the L1 regularization parameter is chosen between [1e-5, 5e-06, 1e-06, 5e-07, 1e-07]. The batch size varies greatly with the amount of data being processed, generally it is chosen by hand from the following [32, 64, 128, 256]. Figure 4.1 shows the Area Under Curve (AUC) for different values in the

Layer	Neurons	Activation function	Regularization
1	64	Sigmoid	-
2	48	ELU	L1
3	32	ELU	L1
4	16	ELU	L1
5	32	ELU	L1
6	48	ELU	L1
7	64	Sigmoid	-

Table 4.1: The used Auto Encoder Model.

grid search. Here it can be seen that the AUC varies greatly and therefore it is important to choose the right parameters

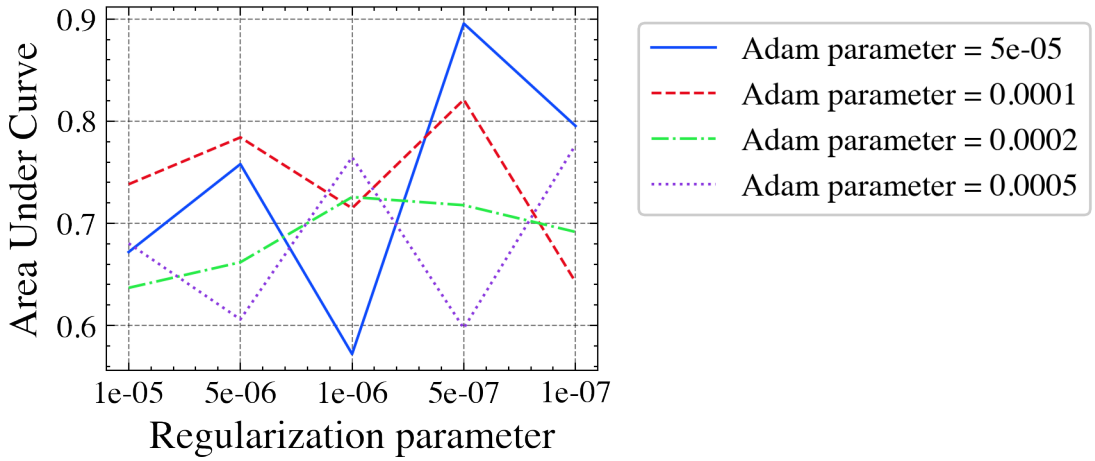


Figure 4.1: Grid search for UNSW-NB15 data

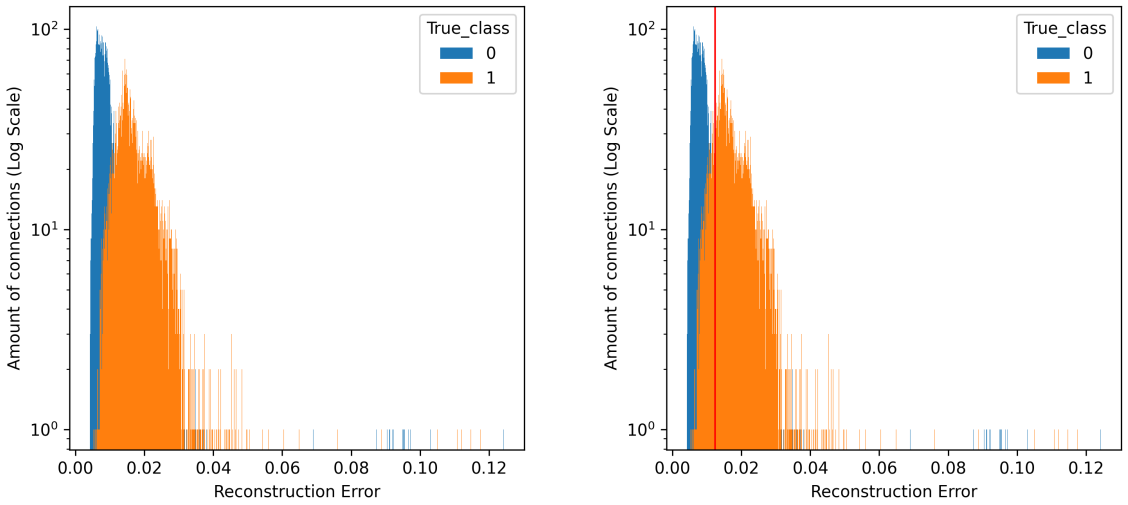
This grid search is executed on the training set of the UNSW-NB15 dataset. Since this is a labeled dataset, metrics such as accuracy and AUC can be used to find the best performing settings. The training set of the CICIDS 2017 dataset contains only benign data. Therefore, it is not possible to use accuracy or AUC as metrics for the grid search. For this problem, solutions are proposed [63] and in related work these parameters are chosen empirically. In this research the histogram of the reconstruction error of the validation split of the training set is used to empirically choose the parameters. A histogram with a low average reconstruction error, but with some outliers to make sure the model is not too general, provides the best results.

Determining the threshold

To determine the threshold for classification of outliers, many different options exist. Yang et al. [64] found in a survey of 100 randomly chosen works on outlier detection that the top measures to automatically choose the threshold are using Standard Deviation, Median Absolute Deviation and Inter Quantile Range. All three of these methods give poor results

with a large number of outliers, as is the case in the two used popular datasets. Moreover, they are sensitive to parameters defined by the researcher.

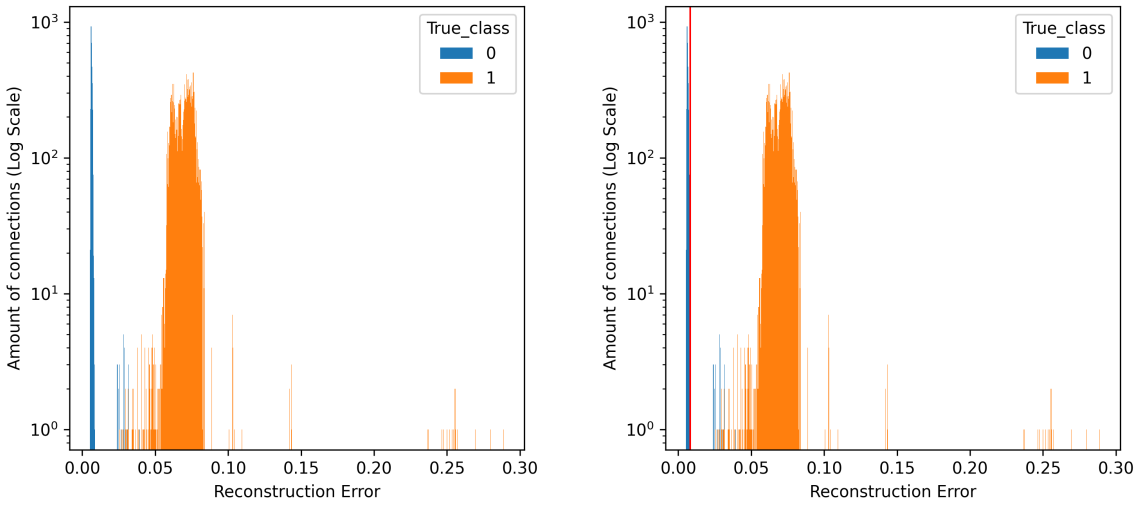
In the experiments conducted in this research the threshold is set manually using the ROC curve and a histogram of the test results. From the histogram, a dip in occurrences is determined to cut off outliers. For example in Figure 4.2a a clear dip can be seen and from this the threshold can be drawn.



(a) Without threshold. *(b) With threshold.*

Figure 4.2: Reconstruction error histogram of HTTP data of the UNSW-NB15 dataset.

To configure the threshold automatically, a custom piece of code has been created which takes the average of a moving window, to disregard sudden jumps in bin size, and searches for series of first decreasing and then increasing moving averages. This indicates a dip in the data, such as in figure 4.2a between the blue and orange peaks. The first increasing moving window value in the longest series of decreasing and increasing values is chosen as the threshold. In figure 4.2b this does not result in the ideal threshold since the moving window size is too large and the first increasing values are disregarded. However, in figure 4.3b the threshold is well chosen, since there is a clearer divide between normal and outlier data.



(a) Without threshold. *(b) With threshold.*

Figure 4.3: Histogram of DNS data reconstruction error

4.3 Improvements on Auto Encoders

Network data is highly dimensional and highly heterogeneous, this makes it difficult for auto encoders, and other deep learning models to learn a good representation of the data and thus be able to identify anomalies. This raises the hypotheses that if the similarity of the data can be improved, then the learned representation can be improved. One way to distinguish different types of network data is to categorize the data per service. In related work, this hypothesis has been approached in several ways.

In 2002, Krugel et al. [65] propose an anomaly detection method with features tailored to specific services. They propose three features to determine an anomaly score:

- Type of Request;
- Length of Request;
- Payload Distribution.

With these three features, they are able to very accurately determine anomalies in DNS traffic. DNS was chosen as since the packets are relatively small. The small packets and small feature size were necessary to find anomalies with a good enough efficiency. This research has been conducted a long time ago and computing power has since increased significantly, making it possible to use larger datasets and feature sets.

In 2011, Sperotto et al. [66] use time series classification and Hidden Markov Models per service to detect attacks in DNS and SSH traffic. The proposed method is based upon the assumption that a network attack follows a certain procedure. For example, for a bruteforce attack, there is first scanning and then a password spray. When those events happen after each other, an attack is observed. However, this is unsuccessful in finding attacks with only a few connections.

In 2020, Labonne et al. [67] split data on internet protocol and thus created separate data streams. The split is obtained by analyzing the packet stream and dividing streams at different levels of the TCP/IP stack. For the research, the 8 streams most prevalent in the CICIDS 2017 dataset are chosen:

Frame, Ethernet, ARP, IP, TCP, UDP, DNS, and HTTP.

By selecting streams at different levels of the TCP/IP stack, it is possible to correlate anomalies in the streams to find anomalous packets, the downside is that the amount of data increases significantly. While the flow-based analysis of the CICIDS 2017 dataset results in 3.119.345 connections, the packet-based analysis results in 56.329.679 packets. When the data is split, this leaves 11.701.690 packets in the largest splits, Frame and Ethernet, which is still much larger than the entire flow-based dataset. Therefore it takes over 12 hours to train the used models.

4.3.1 Our proposed method

To improve the performance of Auto Encoders for unsupervised detection. The performance must be increased by increasing the True Positive Rate and reducing the False Positive Rate

and the efficiency must be increased by reducing the time complexity of the detection.

Using flow-based detection reduces the number of samples on which to train. A flow describes and summarizes connection characteristics therefore a classification can be made on a connection without analyzing individual packets. This reduces the amount of data that needs to be processed and therefore reduces the amount of time needed for classification.

By splitting the data on, for example, on the following service protocols, in the application layer, both the performance and efficiency can be improved:

HTTP, SSH, FTP, DNS, SSL and other

When the data is categorized by only considering data of the same service, malicious data will be more anomalous. This will increase the True Positive Rate and reduce the False Positive Rate. When only considering one service at a time, it becomes possible to create features specific to that service, as proposed by Krugel et al. [65]. With expert knowledge about the services it is possible to create tailored features per service, which will further increase the performance. Using service-specific features is not possible when considering all services together, because a service specific feature might just be indicative of a service. When that service is then a minority in the dataset, the service specific feature will be considered an anomaly.

By splitting the data in different services, there is no overlap created, the data belongs to one category or the other. Therefore, the largest split in the data is the DNS service with 957.812 connections, which is far less than the entire dataset with 3.119.345 connections. It is then much faster to train a model on the DNS data. Since a separate model is created per data split, each separate model can can be trained and used to classify in parallel, the entire data set can then be used for training and testing much faster.

Concluding

Splitting the data per service protocol therefore increases both performance and efficiency in unsupervised auto encoders, regardless of distribution of services in a dataset. Since each service is considered individually, it is possible to select custom services for anomaly detection. For example, the Remote Desktop Protocol (RDP) is often used for attacks, however, when the service is disabled in a network, it is unnecessary to monitor it for anomalies. With knowledge about a network a custom set of services, which are important to monitor can be selected.

4.4 Splitting Data per Service

To evaluate the proposed method of splitting data per service, the UNSW-NB15 and CICIDS 2017 datasets are used. When these datasets are split per service, it gives some splits that are large enough to process as a separate dataset, As can be seen for the UNSW-NB15 dataset in table 4.2. Service 2 through 7 are large enough to serve as a separate dataset, the other services do not hold enough benign connections to create a baseline for normal data. Scikit-learn [68] state that even for simple Machine Learning models 50 samples is the minimum, Deep Learning models such as Auto Encoders generally require more samples. When the smaller services are used separately and the model is trained on, for example, 2 normal connections, the classification only indicates when a new connection is similar to those 2 connections. There might be many more benign connections, which are not similar to those 2 connections. About half of the total amount of data has no associated service and service 8 through 13 are too small to be processed as a separate dataset and will need to be added to the data without a specified service or classified together. Which might still be very dissimilar.

ID	Service	Number of records
1	No Service specified	1.246.397
2	dns	781.668
3	http	206.273
4	ftp-data	125.783
5	smtp	81.645
6	ftp	49.090
7	ssh	47.160
8	pop3	1.533
9	dhcp	172
10	snmp	113
11	ssl	142
12	radius	40
13	irc	31
	Total number of records	2.540.047

Table 4.2: Number of records per service in the UNSW-NB15 dataset

In the sections below, the split in data is discussed for the CICIDS 2017 and UNSW-NB15 datasets, with respect to the distribution of attack and normal data.

4.4.1 CICIDS 2017

Table 4.3 shows the distribution of normal and attack data per service. Since the CICIDS 2017 dataset does not specify individual services, but simply gives the destination ports of a connection, the port numbers with their associated service are shown in the table.

The first thing that can be concluded from table 4.3 is that the attacks are not well distributed per service. For example, DNS, NTP and HTTPS hold almost no attack data and HTTP holds almost all attack data. From this, the mistake can be made that the DNS and NTP protocols can be associated with normal traffic, while the HTTP protocol can be associated with attack traffic. If these services are classified separately, the malicious traffic

will stand out more. The second observation is that for, for example, in FTP traffic there is more attack data than normal data. Since the methods are finding anomalies, when the majority of the data is attack data, the normal data will start to appear as an anomaly. A way to counteract this, is to train the method only on normal data. The third observation is that only the services with the top amounts of data have been included into table 4.3, which means that for a large number of ports it will not be possible to gather enough data to be able to classify them separately. Since there are less than 10 connections to that port and creating a normal baseline will not be meaningful.

Destination Port	Label	Count
21 - FTP	Attack	8181
	Normal	5341
22 - SSH	Attack	6140
	Normal	10801
53 - DNS	Attack	159
	Normal	957812
80 - HTTP	Attack	383239
	Normal	235695
88 - Kerberos	Attack	159
	Normal	5421
123 - NTP	Attack	1
	Normal	23879
137 - NetBIOS	Normal	7917
389 - LDAP	Attack	159
	Normal	6247
443 - SSL	Attack	240
	Normal	505470
564 - SMTP	Attack	160
	Normal	3657

Table 4.3: CICIDS 2017 distribution of normal and attack data destination port

4.4.2 UNSW-NB15

In table 4.4, the distribution of normal and attack data per service is given. The UNSW-NB15 dataset is created, in part, with the use of Zeek [69] logs. Zeek is a network logging application and gives a classification for the used service, these classifications are presented in the table.

Some of the same problems of the CICIDS 2017 dataset also reside in the UNSW-NB15 dataset. The SSH and FTP data services have a very skewed ratio of normal and attack data. This is fixed in the selected training and testing sets. However, the training and testing sets create an unnatural ratio in other services. DNS, for example, now holds far more attack records than normal traffic, making the normal traffic the anomaly when considering the entire training set. This can be alleviated by only selecting normal data for training, just as with the CICIDS 2017 dataset. The next observation is that a number of services, almost, exclusively hold attack data. SSL, DHCP, POP3, IRC and SNMP would be unsuitable for anomaly classification on their own, a possible solution would be to group similar services

together. For example, all services associated with e-mail could be grouped together. Better would be to have sufficient data on each service.

Service	Label	Training set	Testing set	Entire dataset
No service	Normal	36512	27375	1166520
	Attack	57656	19778	79877
dhcp	Attack	94	26	172
dns	Normal	7493	3068	571037
	Attack	39801	18299	210631
ftp	Normal	1218	758	46075
	Attack	2210	794	3015
ftp-data	Normal	2552	949	123893
	Attack	1443	447	1890
http	Normal	5348	4013	187426
	Attack	13376	4274	18847
irc	Normal	0	0	1
	Attack	25	5	30
pop3	Normal	4	0	4
	Attack	1101	423	1529
radius	Normal	2	2	10
	Attack	10	7	30
smtp	Normal	1579	635	76656
	Attack	3479	1216	4989
snmp	Normal	1	0	1
	Attack	79	29	112
ssh	Normal	1291	200	47141
	Attack	11	4	19
ssl	Attack	56	30	142

Table 4.4: UNSW-NB15 distribution of normal and attack data per service

4.5 Capturing Real Data

To show the practical use of the proposed method, a dataset has been created from data captured from a medium-sized network in commercial use. The connections were logged with Zeek [69]. Zeek is an open source network traffic analyzer, used to collect extensive logs of traffic passing through the network. The data has been captured during an entire day, for the dataset, connections with a starting timestamp between 12:00 and 16:00, local time, have been selected. In those 4 hours 4.511.173 connections have been observed.

The distribution of network services in the captured data can be found in table 4.5. This shows, similar to the UNSW-NB15 dataset, that, according to the categories created by Zeek, the "No Service" category is the largest, followed by DNS traffic. The next largest service is SSL, in contrast to the UNSW-NB15 dataset where HTTP is a large service.

The port distribution of the No Service category is shown in table 4.6. This shows that the largest part of the "No Service" category is traffic to port 443, associated with TLS/SSL. The connections are given the "No Service" label, since there is no additional information about the connections in the ssl.log. However, since the connections would show similar characteristics to the connections identified as SSL traffic they could be analyzed together. Or at least separate from other traffic with the "No Service" label, to other ports.

Service	Amount
No Service	2397618
dns	1065914
ssl	713753
krb_tcp	82741
dce_rpc	66574
ntlm,dce_rpc	54712
http	45675
krb,smb,gssapi	18603
dce_rpc,ntlm	18253
gssapi,smb,krb	9351

Table 4.5: Service distribution in the real dataset.

id.resp_p	amount
443.0 - ssl	1757725
389.0 - LDAP	152535
3389.0 - RDP	142428
5985.0 - WINRM	36175
1433.0 - ms-sql-s	26146
41121.0 - tentacle	24723
5246.0 - firewall	19212
80.0 - HTTP	16737
547.0 - DHCP	15086
135.0 - RPC	12435

Table 4.6: Port distribution of the "No Service" category in the real dataset.

Of all Zeek logs, the following six log files will be used:

conn.log, dns.log, ftp.log, http.log, ssh.log and ssl.log

These six log files give valuable information for detection and hold minimal PII information. Using these files useful features can be extracted.

Anonymization

Since the data is captured in a network, which is in commercial use, it contains Personal Identifiable information (PII). For the six selected files, the fields which contain PII are noted in table 4.7

Log file	Field containing PII
conn.log	src IP and dst IP;
dns.log	src IP, dst IP, dns query and dns response;
ftp.log	src IP, dst IP, user, password, arg, src data_channel and dst data_channel;
http.log	src IP, dst IP, host, uri, filename, username and password;
ssh.log	src IP, dst IP and server name;
ssl.log	src ip, dst ip, subject, issuer, client_subject and client_issuer.

Table 4.7: Zeek log files and fields containing PII

The PII fields are then anonymised using three different methods, for different types of data: IP addresses, dns queries and others.

The IP addresses are transformed to random IP addresses. The IP addresses are split in octets and each octet is mapped to a random number between 0 and 256. Which means that IP addresses in each octet are grouped together. For example: 1.2.3.50 and 1.2.3.100 might be transformed to 9.222.51.2 and 9.222.51.244. Thus keeping information about IP addresses originating from the same sub domain.

Something similar is done with dns queries. To be able to preserve information, the domains at each level are mapped to the same random number (label) using scikit-learn's LabelEncoder [70]. For example: google.com and dns.google.com could be mapped to 5.3 and 1.5.3. If a dns query has more than 1 level of sub-domains, the entire sub-domain is mapped to a number. For example, dns.google.com and dns.dns.google.com could be mapped to 1.5.3 and 2.5.3.

The other PII fields are encoded with scikitlearn's LabelEncoder in its entirety. Which means that the same inputs are mapped to the same number. Based on the occurrence of that input, the input which occurs most often, is assigned 0 and so forth.

All transformation tables and LabelEncoders are deleted after the anonymisation process, which makes it irreversible. Some meta-information is kept to be able to extract features for classification, but the data cannot be traced back.

4.6 Feature creation

To build a database for anomaly detection from the collected data. Some features need to be added. As a basis, the features from the UNSW-NB15 [10] have been analyzed, which in turn were inspired by the features of the NSL-KDD [6] dataset. In addition to a selection of the UNSW-NB15 features, some of our own features have been added. The features are divided into three parts: features that can be extracted from Zeek's [69] main connection

log, conn.log; features that can be extracted from other connection logs, http.log, dns.log, ftp.log, ssh.log and ssl.log, and features that are created from statistical analysis over that connection and a number of connections before it.

Number	UNSW-NB15 feature	Corresponding feature	NSL-KDD feature	Corresponding feature
1	sload	24	src_bytes	8
2	sbytes	8	service	6
3	ct_state_ttl	X	dst_bytes	9
4	sttl	16	diff_srv_rate	X
5	smeansz	26	flag	X
6	dur	7	same_srv_rate	X
7	dload	25	dst_host_srv_count	29
8	dintpkt	X	dst_host_same_srv_rate	X
9	dbytes	9	dst_host_diff_srv_rate	X
10	dttl	17	dst_host_serror_rate	X

Table 4.8: Top 10 features per dataset and corresponding new feature, based on Mutual Information

To determine which features from the UNSW-NB15 and NSL-KDD feature set are useful to recreate, the Information Gain or Mutual Information has been calculated. The data is split in two classes, attack data and normal data, and the Mutual Information is obtained by calculating the relative entropy over those classes for each feature, using SciKit-learn's MutualInformation [71]. In table 4.8 the top 10 features from the UNSW-NB15 dataset and the NSL-KDD dataset are shown, together with the number of the feature from our feature set that corresponds to it or an X when the feature has not been added to our feature set. Some features have been left out since they are hard to recreate from Zeek logs, for example dintpkt, the destination inter packet arrival time feature from the UNSW-NB15 dataset. These features are hard to recreate since Zeek logs information about the entire connection and for the inter packet arrival time, information per packet is needed.

The first set of features can be found in table 4.9. Features 0 through 15 are readily available in the standard version of Zeek. Features 16 and 17 are not readily available since Zeek logs the entire connection, there is no information on individual packets. The TTL value in a TCP connection is placed in a SYN packet and is therefore not included in information about the entire connection. For this, a custom script has been created, which records the TTL value in every SYN packet and adds the last seen values to the connection log. This script can be found on Github [72]. Since the event has to be triggered this could cause a performance overhead. To monitor this, the memory and CPU usage have been observed during usage of the extra features and during normal operations. Figure 4.4 shows the free memory on the device running Zeek. The area in grey shows the moment Zeek was restarted with the extra features, the captured logs of that day, before that moment are then archived,

ID	Name	Type
0	Timestamp	Time
1	Source IP address	String
2	Source port number	Integer
3	Destination IP address	String
4	Destination port number	Integer
5	Transaction protocol	String
6	Service	String
7	Duration	Float
8	Source bytes	Integer
9	Destination bytes	Integer
10	State	String
11	Source is local origin	Bool
12	Destination is local origin	Bool
13	Missed bytes	Integer
14	Source packet count	Integer
15	Destination packet count	Integer
16	Source TTL	Integer
17	Destination TTL	Integer

Table 4.9: Features extracted from conn.log

explaining the increased free memory. After the restart, the decrease in free memory is comparable to the situation without recording the features. Figure 4.5 and Figure 4.6 show the system and user CPU usage, respectively. The CPU usage follows an expected period with decreased usage during lunch time and at the end of business. There is no noticeable increase in CPU usage, only difference in the time peaks occur.

The second set of features can be found in table 4.10. Features 24 through 27 can be created from the information of that particular feature, for example, feature 24 is created by dividing feature 8, Source bytes, by feature 7, Duration. Features 28 through 24 are created by a statistical analysis over the last 100 connections per connection. Therefore, these features are cost expensive to create. While most of the code for this project is written in Python code, these operations are written in C code to speed up feature creation. This yielded a reduced feature creation time from 40 minutes to 2:30 minutes, for over 10 million records.

These statistical features provide information on earlier connections and make it that not every connection is weighed individually. For example, a Denial of Service attack, might be recognizable by a short duration, but far more important is the number of connections per 100 connections.

The third set of features can be found in table 4.11. These features are created from

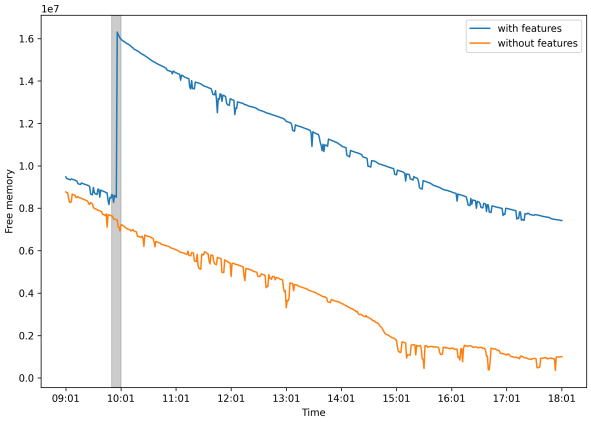


Figure 4.4: Free memory on the device running Zeek, with and without TTL features.

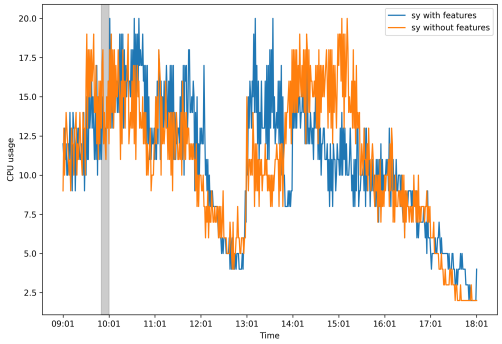


Figure 4.5: CPU usage by the system on the device running Zeek, with and without TTL features.

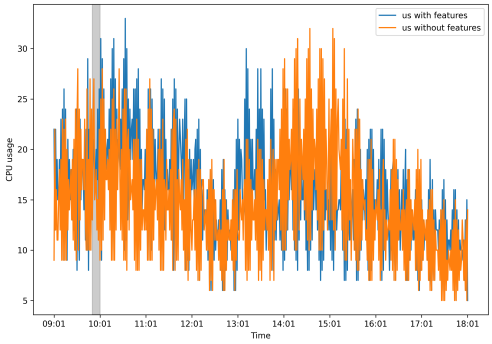


Figure 4.6: CPU usage by the user on the device running Zeek, with and without TTL features.

the other selected connection logs. These logs hold information on the services used in the connection and are 0 if the service is not used. Therefore, there are only a limited amount of connections that receive information on these extra features. These features provide more information on a specific service and therefore give additional information on which to determine whether a connection is an outlier. Features 35-54 hold extra information on DNS, HTTP, SSH and SSL. These features can only be used when classifying the data separated per service, because they are only applied to a limited number of connections. For example, when only a small portion of the connections are SSH connections, the features only present in SSH connections stand out as outliers. While they might just be normal connections. Feature 55 is created from the weird log and is 1 if the connections are present in the weird log. This log file contains information on connections that do not behave as expected. For example, by having malformed packets or headers.

ID	Name	Type
24	Source bytes per second	Float
25	Destination bytes per second	Float
26	Mean Source packet size	Integer
27	Mean Destination packet size	Integer
28	No. of connections that contain the same service (14) and source address (1) in 100 connections	Integer
29	No. of connections that contain the same service (14) and destination address (3) in 100 connections	Integer
30	No. of connections of the same destination address (3) in 100 connections	Integer
31	No. of connections of the same source address (1) in 100 connections	Integer
32	No of connections of the same source address (1) and the destination port (4) in 100 connections	Integer
33	No of connections of the same destination address (3) and the source port (2) in 100 connections	Integer
34	No of connections of the same source (1) and the destination (3) address in in 100 connections	Integer

Table 4.10: Statistical features created from other features presented above

ID	Name	Source log	Type
35	Query type	DNS	Integer
36	Return code	DNS	Integer
37	rtt	DNS	Integer
38	TTLs	DNS	Integer
39	dns_query_len	DNS	Integer
40	method	HTTP	String
41	trans_depth	HTTP	Integer
42	http_query_len	HTTP	Integer
43	status_code	HTTP	Integer
44	referrer_bool	HTTP	Binary
45	auth_success	SSH	Integer
46	auth_attempts	SSH	Integer
47	direction	SSH	Binary
48	client	SSL	String
49	version	SSL	String
50	resumed	SSL	Binary
51	last_alert	SSL	String
52	next_protocol	SSL	String
53	established	SSL	Binary
54	validation_status	SSL	
55	weird_bool	Weird	Binary

Table 4.11: Features created from http.log, dns.log, ssh.log, ssl.log and weird.log

Chapter 5

Evaluation of the Proposed Method on Popular Datasets

In this chapter the answer to research question 2: *Which improvements can be made on Auto Encoders for anomaly-based NIDS?* will be addressed. A method of data preparation has been proposed which improves the performance and efficiency of Auto Encoders for unsupervised anomaly detection. The performance is considered improved if the True Positive Rate can be increased and the False Positive Rate can be decreased, and the efficiency is considered improved if the time complexity can be reduced. To evaluate the proposed method, experiments will be conducted on two popular datasets, discussed in section 3.1. The CICIDS 2017 and UNSW-NB15 dataset will be used to demonstrate that the performance of unsupervised Auto Encoders increases, regardless of the used dataset. It will be discussed and demonstrated how the proposed method will logically increase the efficiency.

5.1 Performance on Popular Datasets

In this section, the improved performance with our presented method for unsupervised anomaly-based intrusion detection is presented. Experiments are conducted on two popular IDS datasets to show that the performance of unsupervised Auto Encoders increases when data is split per service. For all figures shown in this section, the points depict a connection, the x axis the index of that connection and the y axis the reconstruction error of that connection.

5.1.1 CICIDS-2017

Figure 5.1 shows the reconstruction error of all data in the CICIDS 2017 dataset. This shows a large number of false positives. In figure 5.2a only the HTTP data is used for evaluation and this shows a similar detection rate with a much lower false positive rate. Table 5.1 summarizes the classification of data, when all services are evaluated together and the results when the services are evaluated separately. It can be seen that while the detection rate for all data and HTTP data is similar the false positive rate is much lower. When looking at SSH and FTP data, the detection rate also improves and the method is able to find attacks that are not found when analyzing the entire dataset together.

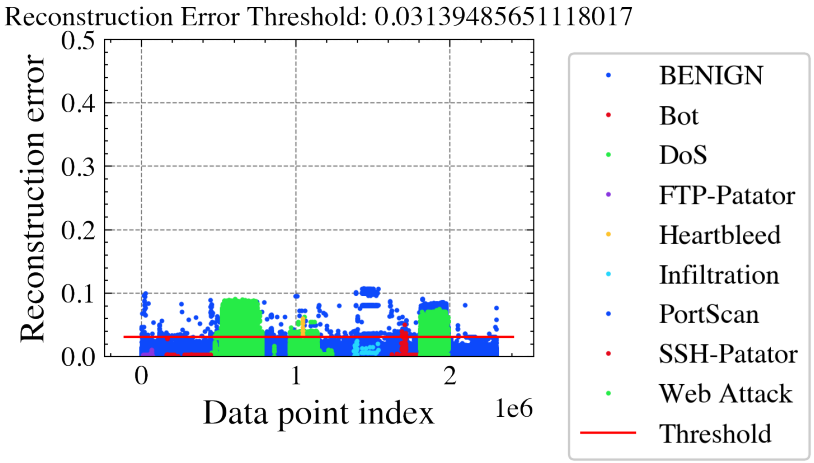
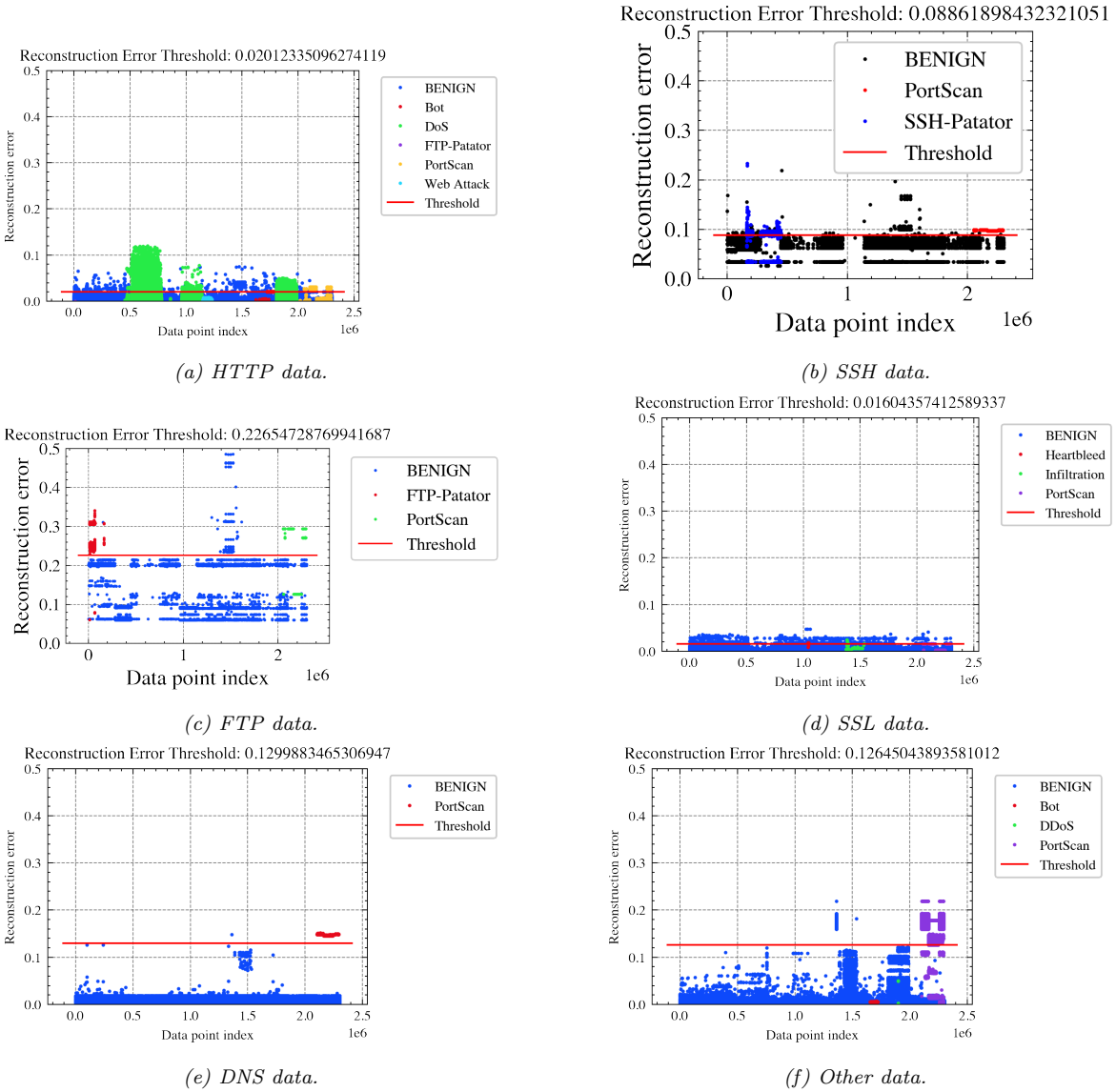


Figure 5.1: Reconstruction error of all services of the CICIDS 2017 dataset together.

Figure 5.2 shows the reconstruction error for all separate services. When looking at the reconstruction errors of the SSH, figure 5.2b, and especially FTP services, figure 5.2c, a pattern can be observed. This pattern is due to the limited amount of training data, therefore the model is not able to create a good general model over all features. The lines of the pattern are created when a single feature is misclassified for all connections. For example, a certain feature has had a value of 0 for all training connections and has a value of 1 for all testing connections. Multiple lines then signal increasing amounts of features which are all classified wrong. A more gradual increase in reconstruction error, such as with the "other services" category, figure 5.2f, shows that the training data holds more diverse values for each feature.



(a) HTTP data. *(b) SSH data.*

(c) FTP data. *(d) SSL data.*

(e) DNS data. *(f) Other data.*

Figure 5.2: Reconstruction error of the CICIDS 2017 dataset

5.1.2 UNSW-NB15

Fig. 5.3 shows the reconstruction error of all data together. This shows that, while the attack data has a higher reconstruction rate. There is still a large number of normal connections with a large reconstruction error and therefore classified as an anomaly.

In Fig. 5.4 the results of classifying just the DNS service are shown, the Auto Encoder model is trained with only normal data, to be able to classify the attack data as anomalies, as discussed in section 4.4.2. As can be seen in Fig. 5.4, the attacks show a much larger reconstruction error than normal data. This indicates that the model learns the normal data well. Even though the ratio of normal and attack data for the DNS service is heavily skewed towards attack data, the malicious data is successfully classified as anomalies.

When looking at a service which has more diverse data, such as the HTTP service or traffic without an associated service, the distinction between normal and attack data is less

Data	All		HTTP		SSH		FTP		DNS		SSL		Other	
	A	T	A	T	A	T	A	T	A	T	A	T	A	T
BENIGN	13242	1743179	2323	186133	242	8669	118	4381	1	743138	1253	364518	950	436279
Bot	11	1966	21	1261									0	705
DoS	170802	380688	212657	380685									0	3
FTP-Patator	0	7938	0	1			7920	7937						
Heartbleed	11	11									9	11		
Infiltration	0	36									7	36		
PortScan	0	158930	188	533	243	243	140	244	159	159	191	449	156300	157302
SSH-Patator	0	5897			2921	5897								
Web Attack	0	2180	0	2180										

Table 5.1: Classification of CICIDS 2017 data for all data and data split per service. A=Anomaly, T=Total

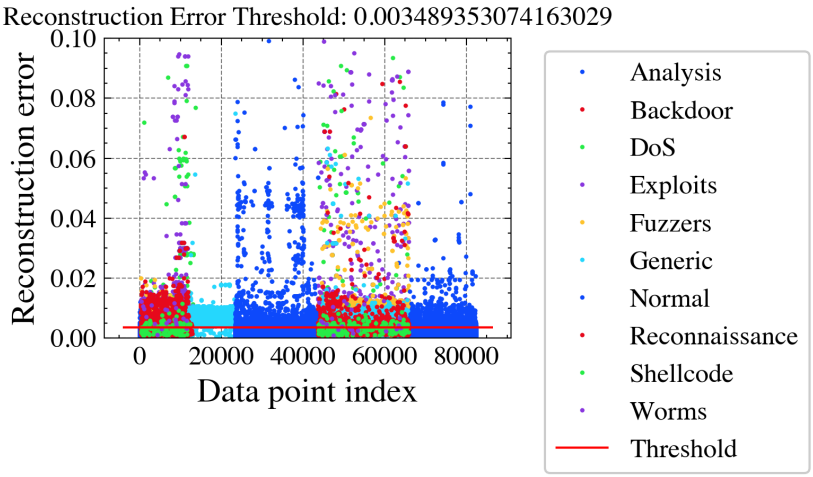


Figure 5.3: Reconstruction error of all data of the UNSW-NB15 dataset.

clear. As can be seen in Fig. 5.5a. This causes the model to classify more normal data as anomalies and thus produce more false positives. Most False Positives are obtained in the "No Service" category as can be seen in figure 5.5b.

Table 5.2 shows the classification results when all services are classified together and when the services are classified separately. This shows that the True Positive Rates are similar, however the False Positive Rate is much lower when each service is classified separately. The majority of the False Positives originate from the "No Service" category. While in the DNS and HTTP services there are very few False Positives. All other services, which were too small to classify separately, have been classified together in the "Other" category. While the results are good in this category, most of the normal data is SMTP and SSH service data and most attacks reside in other services, as discussed in section 4.4. Therefore the model might just classify the distinction between SMTP and SSH and other data in this category. It can be concluded that the model is successful in classifying the attack and normal data in the SMTP and SSH services, since these hold enough normal and attack data.

Data	All		DNS		HTTP		FTP		No Service		OTHER	
	Anomaly	Total	Anomaly	Total	Anomaly	Total	Anomaly	Total	Anomaly	Total	Anomaly	Total
Analysis	643	677							674	677		
Backdoor	559	583			9	9	2	2	557	572		
DoS	3841	4089	40	40	387	493	29	29	3173	3372	146	146
Exploits	10498	11132	68	68	2458	2804	912	941	5419	5839	1468	1468
Fuzzers	3256	6062	17	17	234	251	183	267	3686	5527		
Generic	18748	18871	18162	18162	206	213	2	2	292	411	78	78
Normal	8774	37000	30	3068	352	4013	346	1707	5290	27375	13	837
Reconnaissance	2787	3496	12	12	463	470			1678	2992	22	22
Shellcode	262	378							209	378		
Worms	31	44			34	34			5	10		

Table 5.2: Classification of UNSW-NB15 data for all data and data split per service.

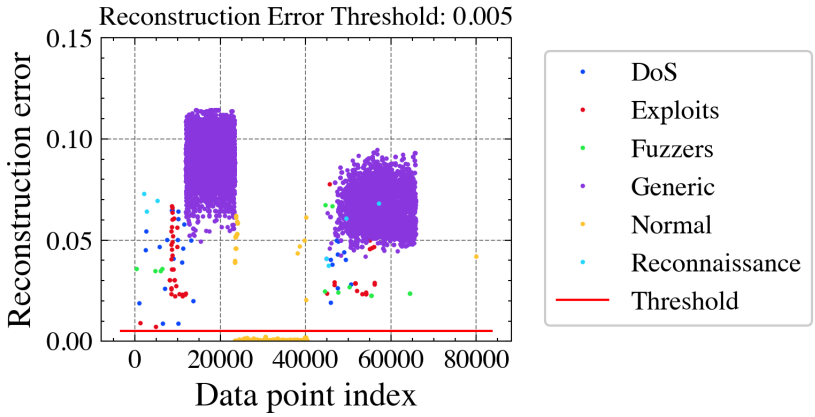


Figure 5.4: Reconstruction error of the DNS service data.

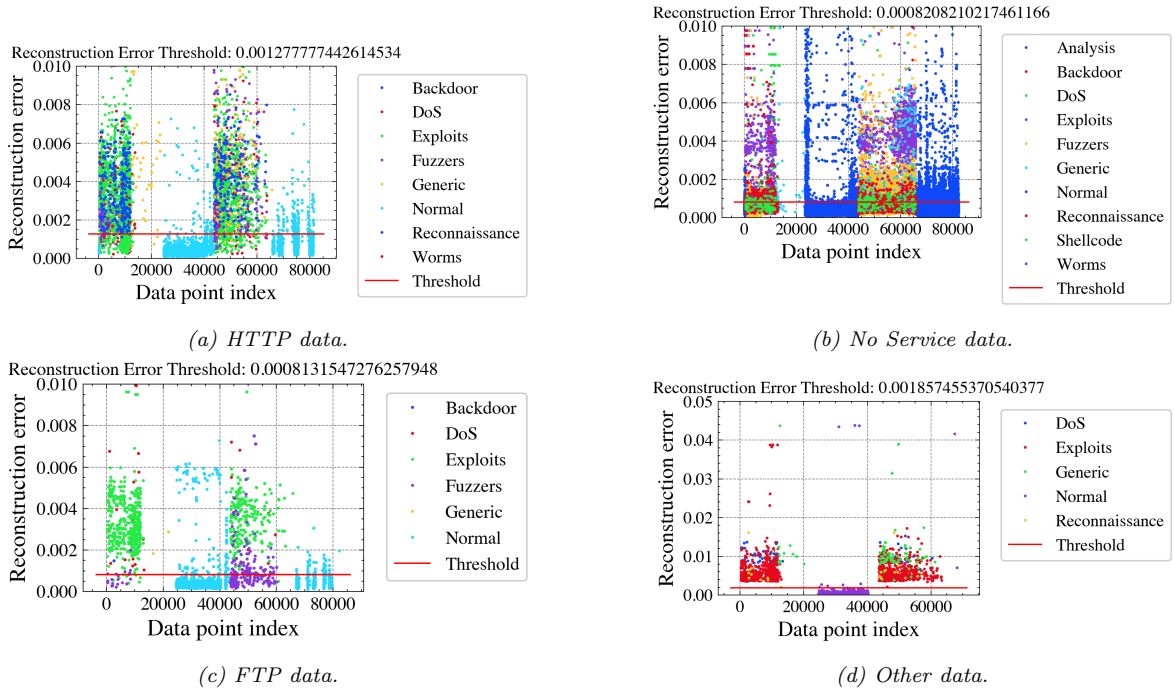
Concluding

Table 5.3 improvement on the True and False Positives rates of the UNSW-NB15 and CICIDS 2017 datasets when all services are classified together and with the proposed method of splitting the data per service. For the UNSW-NB15 datasets this table shows similar True Positive Rates, however there is a clear improvement in the False Positive Rate. For the CICIDS 2017 dataset, there is a clear improvement on both the True Positive Rate and False Positive Rate and table 5.1 shows that with the proposed method, attacks can be found successfully that were not visible when classifying all data together.

	TPR	FPR
UNSW-NB15 All data	0.89616606	0.23713514
UNSW-NB15 Proposed Method	0.896232242	0.159621622
CICIDS 2017 All data	0.3063305	0.0075965
CICIDS 2017 Proposed Method	0.6827916	0.0028036

Table 5.3: Improved True and False Positives Rates using the proposed method.

Table 5.4 places the achieved results of the proposed method in the context of related work, in terms of recall, precision and F1 score. The metrics are compared to researches which



(a) HTTP data. *(b) No Service data.*

(c) FTP data. *(d) Other data.*

Figure 5.5: Reconstruction error of the UNSW-NB15 dataset

used the training and testing set of the UNSW-NB15 dataset. For the UNSW-NB15 dataset the proposed method is performing the best in terms of recall and F1 score, even better than the selected supervised method. Due to the chosen threshold, AE and Local Outlier Factor [73] scores better in terms of precision, however the recall is also much lower. For the CICIDS 2017 dataset, the proposed method does not outperform the related work. The proposed method reaches the second highest precision, after Auto Encoders [74], but does reach a recall which is almost 3 times higher. The supervised method clearly outperforms all other methods in terms of Recall and F1 score on the CICIDS 2017 dataset.

Dataset	Method	Year	Recall	Precision	F1 score	Supervised
UNSW-NB15	ALAD [75]	2020	0.8583	0.8473	0.8527	No
	MR-DHPN [76]	2019	0.862	0.844	0.853	Yes
	AE+LOF [73]	2019	0.7	**0.96**	0.82	No
	This research	2021	**0.8691**	0.8694	**0.8694**	No
CICIDS 2017	ALAD [75]	2020	0.8268	0.8260	0.8264	No
	MR-DHPN [76]	2019	**0.996**	0.986	**0.986**	Yes
	AE [74]	2019	0.2330	**0.9993**	NA	No
	AE+LOF [73]	2019	0.76	0.89	0.82	No
	This research	2021	0.6828	0.9873	0.8073	No

Table 5.4: Results for the UNSW-NB15 and CICIDS 2017 datasets in related work.

5.2 Efficiency on Popular Datasets

The improvement of the proposed method also applies to the efficiency, by reducing the time complexity of the anomaly detection. Table 5.5 shows the time the model needs for training and testing the complete CICIDS 2017 dataset, the DNS data and SSL data with 200 epochs training. The table shows that the time decreases significantly when only the DNS data is used for training, from 10 minutes and 34 seconds to 4 minutes 45 seconds. Further decrease of the training time is shown where the training time for SSL data is 3 minutes and 5 seconds. This follows logically, since there is less data, it takes less time to train. Table 5.5 also shows the time it takes to test the different data streams. This shows that the testing time is significantly less than the training time.

Type of Data	Action	Number of connections	Time (no early stopping - 200 epochs)	Time (with early stopping)
All Data	Training	529.918	0:10:34.25	0:02:59.09
	Testing	2.300.825	0:00:51.14	0:00:51.14
DNS	Training	214.674	0:04:45.76	0:01:12.82
	Testing	743.297	0:00:17.00	0:00:17.00
SSL	Training	140.952	0:03:05.35	0:01:32.72
	Testing	365.119	0:00:05.20	0:00:06.79

Table 5.5: Training and testing times for the CICIDS 2017 dataset in h:mm:ss

The parameter of 200 epochs training is chosen by observing the results of the training and at this time the model almost does not improve, as can be observed in figure 5.6. However, similar results can be obtained with much less training. For this, early stopping is used. The early stopping criterion is that the decrease in reconstruction error on the validation set is lower than 0.00001 for 5 epochs, when that occurs the model stops training. This means that when a model is trained until the data is learned well enough that there is little improvement, as shown in figure 5.7. Instead of 200 epochs, only 64 epochs are needed to reach a sufficiently low reconstruction error.

Table 5.5 shows the comparison of training times when early stopping is used. What is interesting to note is that the training time of the DNS dataset is now lower than the training time of the SSL dataset, even though it is trained with more data. This is due to the fact that the DNS data is more similar and therefore the model is able to quicker reach a low reconstruction error.

Figures 5.8 and 5.9 show the increase in training time for smaller and larger training sets, respectively. This follows a linear increase in time with an increase in training size. The connection is not completely clear, this is due to the model weights being initialized randomly. This might lead to a quicker low reconstruction error is some cases.

In figure 5.10 the time needed for testing with increasing amounts of testing data is shown, this shows a clear linear connection since there is no randomness involved in testing the data.

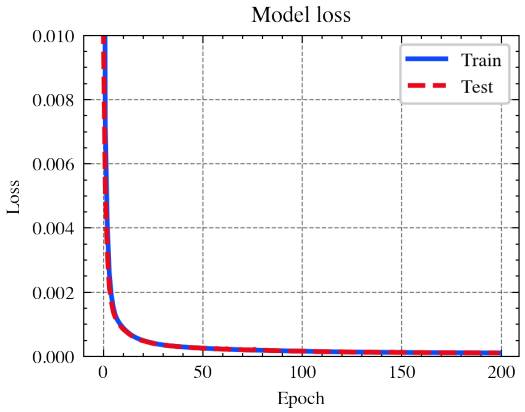


Figure 5.6: Reconstruction error for 200 epochs for all CICIDS 2017 data

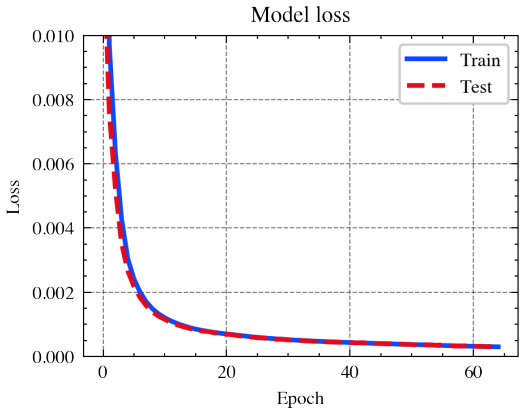


Figure 5.7: Reconstruction error with early stopping for all CICIDS 2017 data

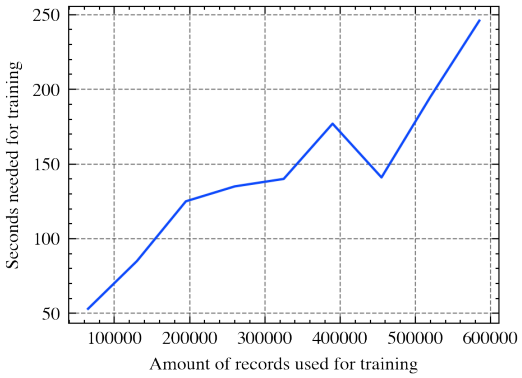


Figure 5.8: Training times for all data in the CICIDS 2017 dataset

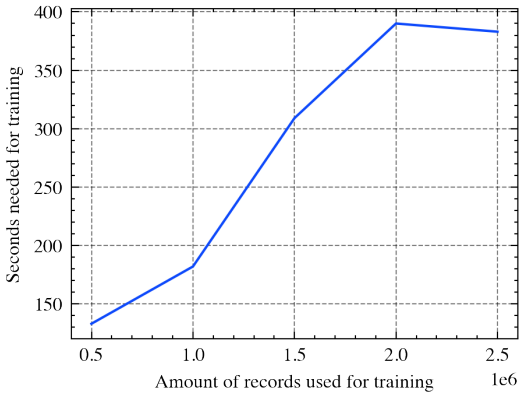


Figure 5.9: Training times for all data in the CICIDS 2017 dataset

Concluding

In this section we have demonstrated the increase in efficiency of the proposed method, by demonstrating the reduction in time complexity when data is split per service. We have shown that there is a linear correlation in the training time and the amount of data for training, meaning that Auto Encoders are more suitable for large amounts of training data than models with a quadratic time complexity. This section also demonstrates that the testing time is far smaller than the training time.

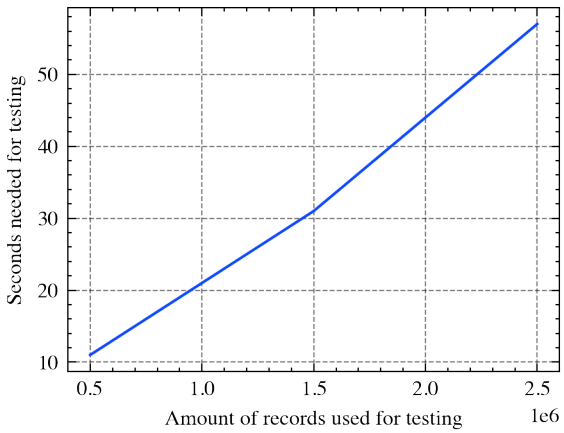


Figure 5.10: Testing times for all data in the CICIDS 2017 dataset

5.3 Conclusions on Improvement for Auto Encoders

In this chapter the answer to research question 2: *Which improvements can be made on Auto Encoders for anomaly-based NIDS?* has been addressed. In this chapter is has been shown that the performance and efficiency of Auto Encoders can be improved by splitting network data on the service protocol in the application layer. The True Postive Rate is improved from 0.30 to 0.68 on the CICIDS 2017 dataset and the False Positive Rate is reduced from 0.0075 to 0.0028 on that dataset. On the UNSW-NB15 dataset the True Positive Rate is similar, however the False Positive Rate is reduced from 0.24 to 0.16. The time needed for training is reduced from 2:59 to 1:12 for the largest split and since the splits can be trained in parallel, this could halve the training time.

Chapter 6

Evaluation of the Proposed Method on Real Data

In this chapter the answer to research question 3: *Can we achieve unsupervised anomaly-based NIDS for practical use?* will be discussed. The method proposed in section 4.3 will be evaluated on data captured in a commercial network to determine whether it is suitable for practical use. An experiment will be conducted on the captured real data to demonstrate the practical use of the proposed method with unsupervised Auto Encoders. The practical use of the proposed method will be discussed and it will be demonstrated that the method is viable for practical use in terms of both performance and efficiency.

6.1 Performance on Real Data

To evaluate the performance of the proposed method. The data has been split into a training and testing set. The first 1 million connections, which correspond to a little over 30 minutes, are used for the training set, the other 3.8 million connections, which correspond to little over 3 hours, are the testing set.

The captured real dataset is unlabeled and expected to only contain benign data, since there were no attacks detected in this time frame by signature-based methods. Figure 6.1 shows the reconstruction error for DNS traffic in the training set of the real dataset. The threshold is based on the histogram and 0.15 percent of the data is reconstructed above the threshold. This indicates that the model can reconstruct the data very well and that there are some clear outliers in the data, which is to be expected in a production network.

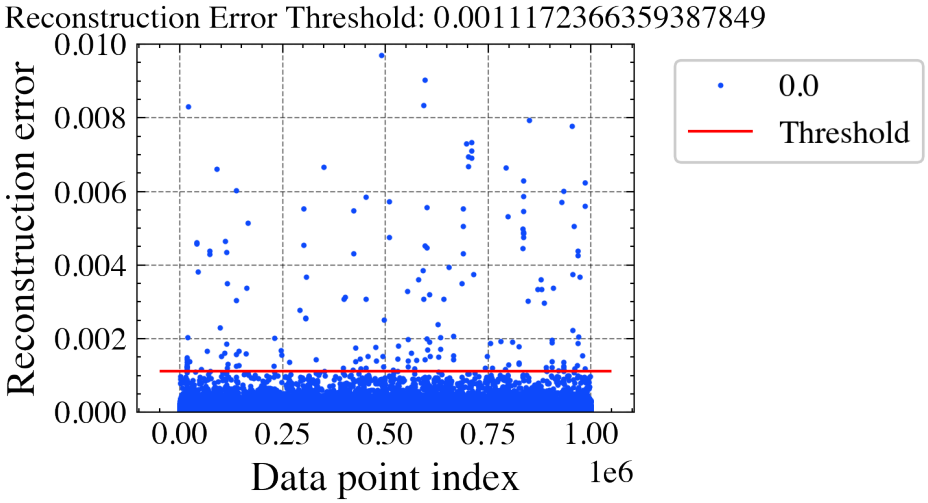


Figure 6.1: Reconstruction error DNS data in the captured dataset

To be able to verify whether the proposed method could detect attacks, some attacks have been inserted into the data. This has been done by taking logs from the UNSW-NB15 dataset that correspond to attacks. These logs are ingested with the normal data and the statistical features are created after the logs are combined as to make it seem like the connections happened in the network. To verify that our method is not only classifying the UNSW logs as anomalies instead of only classifying the attacks as anomalies, we have also inserted benign data from the UNSW dataset as a baseline. Figure 6.2 shows the reconstruction error of the testing set with the inserted logs and table 6.1 shows the amount of connections classified as anomalies and the detection rates. In figure 6.2 it can be seen that the benign data from the UNSW-NB15 data set is reconstructed with a low error, which means that it behaves similarly to the normal data in the real dataset. The attack data of the UNSW-NB15 dataset is reconstructed with a much higher error, table 6.1 confirms this and shows that 54 percent of the attacks are classified as anomalies, while 0 percent of the benign data is classified as an anomaly. Of the data from the real dataset, 0.15 percent is classified as an anomaly.

For the HTTP data, the division is less clear. Figure 6.3 shows the reconstruction error

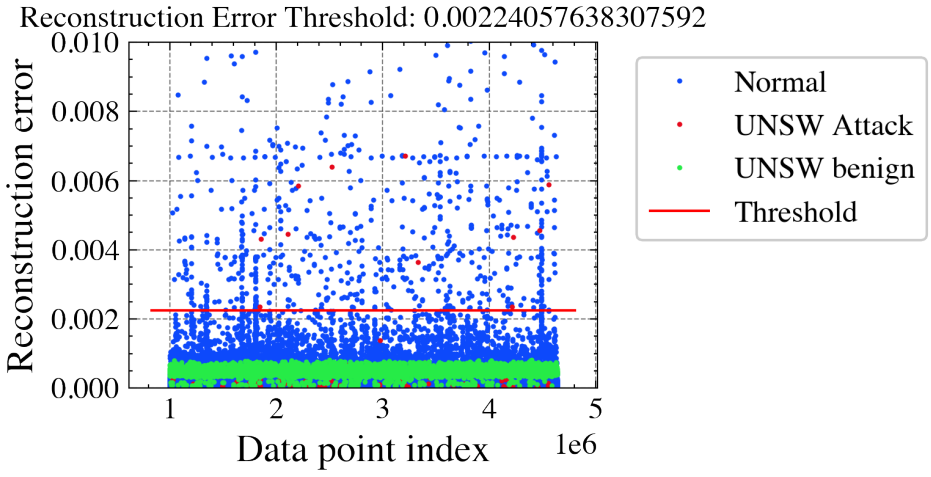


Figure 6.2: Reconstruction error DNS data in the captured dataset

True Class	Anomaly	Total	DR
Normal	1195	835383	0.001430
UNSW Attack	80	148	0.540541
UNSW benign	0	14009	0.0

Table 6.1: Anomalies found in DNS connections of the real data testing set

of the HTTP data in the testing set of the real data set and table 6.2 shows the number of connections classified as anomalies for each class. From figure 6.3 it can be seen that the attack data from the UNSW-NB15 dataset has a higher reconstruction error than the benign data and the data from the real dataset. Table 6.2 confirms this and shows that 81 percent of the attack data is classified as an anomaly while only 27 percent of the benign data is classified as an anomaly. While this division is less clear than in the DNS data, it still confirms that the UNSW-NB15 data is not simply classified as anomalies.

True Class	Anomaly	Total	DR
Normal	1142	36145	0.031595
UNSW Attack	4129	5073	0.813917
UNSW benign	2036	7490	0.271829

Table 6.2: Anomalies found in HTTP connections of the real data testing set

In the HTTP data, 3 percent of data from the real dataset is classified as anomalous, as can be seen in table 6.2. This is significantly higher than the anomalies found in the DNS data. To determine whether such a False Positive Rate is feasible in a practical setting, the cause of the anomalies has been investigated. In table 6.3 the source IP addresses are shown which occurred as anomalies 10 times or more in the test set. IP address 1 through 7 are inserted from the UNSW-NB15 dataset and are therefore not anonymized, the other IP addresses are captured in a commercial network and to preserve privacy they are anonymized in this table. IP address 8, 9 and 10 have been checked with the owner of the network and

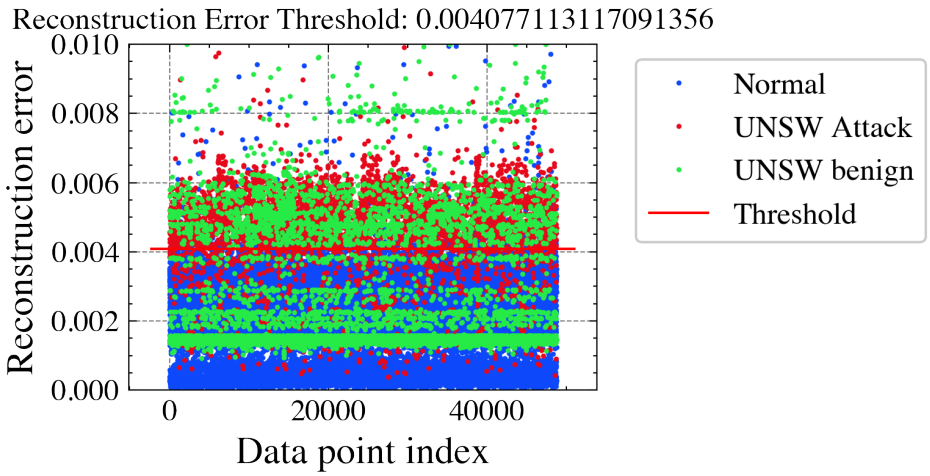


Figure 6.3: Reconstruction error HTTP data in the captured dataset with UNSW attacks and normal data

it has been found that the anomalous behavior of these addresses is to be expected.

The known anomalies can be disregarded and excluded from future detections, which means that there remain, aside form the UNSW-NB15 IP addresses, three IP addresses to investigate. IP address 12 is a known malicious IP, used for scanning and web attacks. IP address 11 or 13 could not be confirmed with the owner of the network, nor were known malicious IP addresses and therefore might be considered as False Positives. It is completely feasible for a security specialist to investigate 3 IP addresses in a little more than 3 hours. Which means that while the False Positive rate of 3 percent is higher than for DNS traffic, it is still feasible to investigate the alarms this detection creates with the chosen threshold of 10 anomalies. If the threshold is lowered, there would be more to investigate and there would be more false positives. Therefore this threshold can be used to tune the detection.

ID	IP Address	Anomaly Count	Anonymized	Cause
1	175.45.176.2	1374	No	UNSW attack
2	175.45.176.3	1100	No	UNSW attack
3	175.45.176.0	839	No	UNSW attack
4	175.45.176.1	816	No	UNSW attack
5	59.166.0.5	723	No	UNSW benign
6	59.166.0.7	665	No	UNSW benign
7	59.166.0.8	648	No	UNSW benign
8	134.32.95.87	225	Yes	Known anomaly
9	134.32.95.115	130	Yes	Known anomaly
10	123.167.67.120	86	Yes	Known anomaly
11	123.167.67.222	30	Yes	No information
12	13.139.58.167	20	Yes	Known Malicious IP
13	247.209.145.27	11	Yes	No information

Table 6.3: Source IP addresses classified as anomalies in the test set of the HTTP data

6.2 Efficiency of the real data setup

To determine whether the proposed method is suitable for practical implementation, the efficiency must be considered. In packet based intrusion detection, there is no need to process the network logs, since the classification is done on the packets themselves. In connection- or flow-based intrusion detection, it is necessary to first process the connection logs into data, which can be used for classification. With the proposed features per method, it is also necessary to combine the information of multiple network logs. Then ultimately the statistical features need to be created. The process can be viewed in table 6.4. The entire process takes 13 minutes and 22 seconds for 10.866.099 connections, which is the amount that has been captured in a little more than 14 hours. Step 1 and 2, reading and combining the logs, are written in Python and not optimized. By writing this part of the process in C, the efficiency could be improved. Step 3 is written in python, but by making use of the underlying C engine, this reduces the processing time of a sample dataset from 40 minutes to a little over 2 minutes.

ID	Step	Time	Total time elapsed
1	Reading logs	04:27	04:27
2	Combining logs	03:11	07:38
3	Creating statistical features	04:48	13:22

Table 6.4: Data preparation time for 10.886.099 records in mm:ss

After all features are created, categorical features need to be one hot encoded and continuous features need to be normalized for training and testing. Table 6.5 shows the amount of time needed to normalize the training and testing data. For 1.000.000 training records the time for normalization is slightly shorter than for normalizing the DNS data, this is due to the service specific features that are created for DNS. This increases the number of features from 35 to 64. The time needed for normalizing the entire testing set is 22 seconds and normalizing the 5 minute test window takes less than a second.

Amount	Type	Time
1.000.000	All - Training	0:00:05.975458
412.674	DNS - Training	0:00:06.499491
3.636.051	All - Testing - 3.5 hours	0:00:22.538049
104.732	All - Testing - 5 minutes	0:00:00.729722

Table 6.5: Time for normalizing real data for training and testing

Table 6.6 shows the training and testing time on the real dataset with early stopping on the entire testing set. This shows that the efficiency is clearly improved by splitting the data per service, since the largest split, the DNS data, takes significantly less time to train than the entire dataset. As described in section 5.2 for the UNSW-NB15 and CICIDS 2017 datasets, the SSL and DNS data take similar amounts of time to train. This is because the DNS data is more similar and therefore easier to reconstruct. The training time on port 3389

data, associated with the RDP protocol, shows that smaller splits without service-specific features also take less time to train. The port 443 traffic, associated with SSL, is more than 3 times as large as the SSL data, however since the service specific features cannot be used, the training time is much shorter than 3 times the training time of the SSL data.

The time needed for testing shows a linear correlation with the amount of data, as discussed in section 5.2. For all data, it takes 44 seconds to classify the testing set. To demonstrate the linear connection, the amount of DNS data captured in a span of 5 minutes is shown in table 6.6, testing this amount of data costs less than half a second.

Type of Data	Action	Amount of connections	Time (with early stopping)
All Data	Training	776.371	0:05:50.982308
	Testing	3.636.051	0:00:44.226864
DNS	Training	184.424	0:01:29.271912
	Testing - Total	849.540	0:00:14.300020
	Testing - 5 minutes	24.317	0:00:00.452973
SSL	Training	120.137	0:01:31.562166
	Testing	563611	0:00:14.300020
Port 3389 - RDP	Training	26.609	0:00:37.991699
	Testing	113191	0:00:01.741728
Port 443 - (SSL/TLS)	Training	412.674	0:02:17.313138
	Testing	1.941.678	0:00:31.462586

Table 6.6: Training and testing times for the real captured dataset in h:mm:ss

Concluding

In this section the efficiency of the dataset creation has been shown. It has been demonstrated that 14 hours of network data can be processed in under 14 minutes. The time needed for normalized data is efficient enough. The time needed for training and testing is low enough to train data in a window of 1 hour and test data in a window of 5 minutes. With this, the efficiency is sufficient for practical implementation.

6.3 Conclusions on the Use of Real Data

In this chapter the answer to research question 3: *Can we achieve unsupervised anomaly-based NIDS for practical use?* has been discussed. An experiment has been conducted on a data captured in a commercial NIDS. It has been shown that using the proposed method and a threshold of 10 anomalies per source IP over almost 3 hours, resulted in 13 detected source IPs of which 4 were UNSW-NB15 attack IPs, 3 were UNSW-NB15 benign IPs, 3 were known anomalies in the network, 1 was a known malicious IP, not detected by signature based detection, and 2 IPs could not be retraced and can be considered False Positives. 12 source IPs are few enough to be investigated in 3 hours by a security specialist and is therefore suitable for practical use. The achieved efficiency is shown to be sufficient to train data in a 1 hour window and test in 5 minute windows and is therefore suitable for practical use.

Chapter 7

Discussion, Conclusions and Future Work

In this chapter the results and methodology of this research will be discussed, the answers to the research questions will be given and a suggestion for future work will be given. A suggestion for the practical implementation of the proposed method will be presented.

7.1 Practical Use

In this section, a proposal for the practical implementation of the proposed method will be given.

7.1.1 Practical considerations

In a practical setting, any alarm which comes in is evaluated by a human security specialist. The task of the security specialist is to determine whether the alarm is a true or false positive and the extent of the impact of the attack. This means that when the number of false positives determines the efficiency of the security specialist. When there are a large number of false positives, the security specialist has less time per alarm to analyze and determine whether it is a true positive. Therefore, if the load of false positives is very high, there will not be enough time to analyze every alarm thoroughly and thus some true positive alarms might be classified as false positives and therefore will go undetected or the extent of the attack is not fully understood.

To improve this process, the number of false positives can be reduced, but also the time it takes to determine whether an alarm is a true or false positive and the extent of the attack can be reduced. The latter can be done by providing more information on a possible attack. Labonne et al. [67] state the following: *Collecting as much information as possible about the attack is also a very important feature for an IDS. This information can then be used to stop or to block the ongoing attack.* For example, when an alarm contains just the information *there is an attack*, it will take a lot of time to analyze every possibility and determine whether there is an actual attack. Alternatively, when the alarm contains the information *there is a port scan from IP address X*, it is fairly easy to confirm whether there is an actual port scan or if the alarm is caused by benign behavior.

7.1.2 Creating an alarm

Since the data is analyzed per service, the detections are based on what happens in a service. For example, when detecting anomalies in DNS traffic.

When the alarm states that anomalous DNS traffic has been detected, it gives already much more information than stating that an anomaly, regardless of the service, has been detected. This gives the security specialist an idea for which type of attacks to look for.

Reconstruction error profile

The reconstruction error provides more information. Since the reconstruction error per feature is known certain properties of the attack become visible. In figure 7.1 the absolute reconstruction error is shown, this shows that Analysis and Backdoor attacks have a high reconstruction error on the features shown in this figure. The absolute values are presented to give a clear overview of how much reconstruction error there is. However, a more positive

True_class	ct_dst_sport_ltm	ct_dst_src_ltm	ct_src_dport_ltm	ct_src_ltm	ct_state_ttl
Analysis	0.011221	0.010807	0.009890	0.008387	0.025543
Backdoor	0.013677	0.011601	0.011014	0.007881	0.026314
DoS	0.010948	0.001005	0.005073	0.006844	0.018352
Exploits	0.007637	0.002792	0.001776	0.006750	0.029392
Fuzzers	0.005025	0.006515	0.003913	0.003369	0.003030
Generic	0.006673	0.003943	0.000175	0.004081	0.031732
Normal	0.002845	0.000745	0.000407	0.003157	0.000066
Reconnaissance	0.004788	0.000052	0.002485	0.004504	0.003983
Shellcode	0.003631	0.001525	0.002575	0.003502	0.000191
Worms	0.002153	0.008249	0.005144	0.004006	0.000147

Figure 7.1: Absolute Reconstruction error per class over all UNSW-NB15 data.

or negative reconstruction error can give more information. For example, a negative reconstruction error on the feature duration gives the indication that the connections are shorter than expected. Which might indicate a network scan.

This information can be provided to the security specialist to help in determining whether the alarm is an attack. By providing the most anomalous features per alarm, a security specialist will know better what to analyze.

7.1.3 Attacks per service

As can be seen in table 7.1, different services are prone to different attacks. By providing the security specialist with the service for which anomalous data has been found, the features which have the most reconstruction error and are thus the most anomalous, and the types of attacks associated with the service. The security specialist will have much more information on the anomaly and will therefore better be able to judge whether an alarm constitutes an attack.

Service	DNS	SSH	FTP
Attacks	NXDOMAIN attack DoS Scanning Exploits Attacks using malformed queries	Bruteforce Credential abuse	Anonymous authentication Directory attack

Table 7.1: Attacks associated with services.

7.1.4 Further improvements

When alarms come in to be analyzed by a security specialist. The security specialist determines whether the alarm is a true or false positive. In doing so, the security specialist is creating a labeled dataset. In related work [22], it has been shown that using a semi-supervised set-up can increase the performance of Auto Encoder methods. The labeled dataset created by the security specialist can then be used for semi-supervised detection for future attacks.

7.2 Discussion

One of the important assumptions that is made in this research, is that the majority of network connections are benign. This assumption makes it possible to train an Auto Encoder on normal data. When the assumption does not hold and a majority or even a large portion of the data on which the Auto Encoder is trained, is attack data. The Auto Encoder will learn the attack data as normal and therefore those attacks will go unnoticed in new data. This could possibly be used by adversaries, to inject low impact attack data in the training set, so that an high impact attack in the testing set will go unnoticed.

From the literature study and the preliminary evaluation, Auto Encoders were indicated to be viable for commercial production in terms of performance and efficiency. Therefore, in the remainder of the research, Auto Encoders have been used for testing and evaluating the results of splitting data per service. It is possible that other methods perform better on the split data than Auto Encoders. Support Vector Machines (SVM) were indicated to have good performance, but are lacking in efficiency. When classifying services with less data, for example, SSH, the efficiency of SVM might be sufficient and the performance might be better than Auto Encoders. Due to the scope and time constraints of this research, this has not been tested.

Auto Encoders have been found to be the most viable method for practical implementation from the literature study conducted in chapter 2. For the demonstration of the improved performance and efficiency of the proposed method, the used Auto Encoder model has been sufficient. However, in related work there are good results shown by Stacked Auto Encoders, Variational Auto Encoders or highly specialized clustering methods that might outperform the Auto Encoder model used in this research.

To make the proposed solution practical, it is necessary that it can be executed in real-time. We have shown that the efficiency of our method is high enough to be executed in training batches of 1 hour and testing batches of 5 minutes, with the amount of data that a medium sized commercial network produces. Since almost all code is written in Python, for easier implementation, the process is far from optimized. When the method is refactored in C code, there is still room for improvement in efficiency.

Thresholds for anomaly detection are often chosen visually in related work. In this research an attempt to choose the threshold automatically has been made. The main difficulty with this is knowing the number of anomalies. In the popular datasets some of the data splits hold more malicious than normal data, this requires a different thresholding strategy than when only a small amount of data is anomalous. Therefore, the thresholding strategy needs to be developed further, due to time constraints and the scope of this research this has not been fully researched.

Tuning parameters in a supervised setting is done using metrics such as Area Under Curve (AUC), by having a labeled training set, the optimal parameter can be chosen. In an unsupervised setting this is not possible. In academic works, there are suggestions on the implementation of unsupervised parameter tuning. In this research, the parameters are tuned visually using a histogram of the reconstruction error. However, since the AUC metrics proved

to be susceptible to changing parameters, it will be worthwhile to automate and optimize the parameter tuning process.

Each connection was separately determined as an anomaly or normal data. With this approach, attacks with only a small number of connections can be detected. The downside is that attacks often consist of a large number of connections and therefore it might be needed to correlate these connections together to find an attack.

To evaluate the proposed method, data from a commercial network has been collected. The signature-based detection showed no attacks on the network in that day. This gives no guarantee, but an indication that there were no attacks. This was done to create a normal baseline on which to train and validate the method. The attacks injected into the normal traffic did not provide the most representative results, therefore it will be better to monitor the network for a longer period to capture actual attacks in the network. This can then be used for more representative results.

In related work, the combination of using Auto Encoders with other methods showed promising results, due to the scope of this research and the time constraints this option has not been fully explored.

The performance of the proposed method on the popular datasets show a clear improvement for some services, mainly DNS and HTTP. However, deciding which data is split together proved vital. In the classification of the CICIDS 2017 dataset, the Heartbleed data was grouped with "Other data" since the connections had port 444 instead of port 443 as destination. Then the detection rate of the Heartbleed attack decreased from 1 to 0. Which means that it was detected when training on all data and it was not detected when the data was split per service. One explanation for this is that since Heartbleed is an SSL attack, it is better detected when it is weighed against SSL data. When the SSL split contains both port 443 and 444. The Heartbleed attack is detected again.

From the results presented for the classification of the CICIDS 2017 dataset it can clearly be seen that DoS and scanning attacks are well detected, since DoS has a detection rate of 0.559 and scanning has a detection rate of 0,994. While web attacks and botnet traffic go undetected. An explanation for this is that the traffic for these attacks is very similar to benign traffic. Whereas in DoS and scanning, traffic characteristic such as frequency and duration are different from benign traffic. By introducing service specific features, such as query length, might improve the detection of attack similar to benign traffic.

7.3 Conclusions

To guide this research, the following three research questions have been formulated and answered:

7.3.1 RQ 1: Are Auto-Encoders the most viable method for practical use in the state-of-the-art of anomaly-based NIDS?

In chapter 2 we have presented an overview of the state-of-the-art in anomaly-based NIDS. By discussing relevant datasets and methods presented in academic works. From the conducted literature study Auto Encoders have been selected as a viable method for practical use, in terms of both performance and efficiency. Supervised methods, such as Decision Trees, presented the best results. However, in a practical setting, it is unfeasible to use labelled data. Therefore Auto Encoders have been chosen as the best performing unsupervised method. With F1-scores of over 0.90 and sufficient efficiency for classification in batches.

7.3.2 RQ 2: What improvements can be made on Auto Encoders for anomaly-based NIDS?

In chapter 4 we have presented our methodology for using Auto Encoders and the proposed method for data processing for the improvement of Auto Encoders. By splitting network data per service, the data becomes more similar and anomalies stand out better. We have shown that the True Positive Rate is improved from 0.30 to 0.68 on the CICIDS 2017 dataset and the False Positive Rate is reduced from 0.0075 to 0.0028 on that dataset. On the UNSW-NB15 dataset the True Positive Rate is similar, however the False Positive Rate is reduced from 0.24 to 0.16. The time needed for training is reduced from 2:59 to 1:12 for the largest split and since the splits can be trained in parallel, this could halve the training time. Therefore Auto Encoders can be improved if the network data is first split per service protocol.

7.3.3 RQ 3: How can we achieve unsupervised anomaly-based NIDS for practical use?

In chapter 4 a dataset has been presented from data captured in a commercial NIDS. In this dataset, service-specific features, such as DNS query length, have been proposed to further increase the performance of the proposed method. It has been shown that using the proposed method and a threshold of 10 detections per source IP over almost 3 hours, this resulted in 13 detected source IPs of which 4 were UNSW-NB15 attack IPs, 3 were UNSW-NB15 benign IPs, 3 were known anomalies in the network, 1 was a known malicious IP, not detected by signature based detection and 2 IPs could not be retraced and can be considered False Positives. 13 source IPs are few enough to be investigated in 3 hours by a security specialist and the performance is therefore suitable for practical use. The achieved efficiency is sufficient to train data in a 1 hour window and test in 5 minute windows and is therefore suitable for

practical use. By splitting the data per service protocol, unsupervised anomaly-based NIDS can be used for practical implementation.

7.3.4 Contributions

The academic contribution is a method for data processing that improves the results of Auto Encoders for unsupervised anomaly-based NIDS regardless of the used dataset, in terms of performance and efficiency. By increasing the True Positive Rate, reducing the False Positive Rate and reducing the time complexity of the classification.

The engineering contribution is a Proof of Concept for the proposed method on Zeek logs for unsupervised anomaly-based NIDS using Auto Encoders.

7.4 Future Work

The implemented Auto Encoder model is sufficient for the method presented in this paper. However, in related work, improved performance is shown using Variational Auto Encoders.

The current use of auto encoders classifies each connection separately. However, in many attacks a large number of connections are used. It would therefore be beneficial to be able to take into consideration what the classifications of related connections are. This is done to some degree by using statistical features that provide information on the related connections. A more in-depth correlation using LSTM Auto encoders or Hidden Markov Models could improve performance.

In related work hybrid models, using Auto Encoders and Decision Trees or Support Vector Machines show improved performance in respect to using only Auto Encoders. As described in section 7.1, the labeled data generated by a human security specialist could be used for semi-supervised hybrid detection using Auto Encoders and Decision Trees.

The data captured in a commercial NIDS did not contain any attacks that were detected using signature-based detection. In future work it will be interesting to run the experiment for a longer time to determine whether the attacks detected by signature-based detection are also detected by the proposed method. Similarly, the number of False Positives in comparison with the signature-based detections can be researched to determine the extent to which the proposed method can be implemented.

Bibliography

[1] A. Aldweesh, A. Derhab, and A. Z. Emam, "Deep learning approaches for anomaly-based intrusion detection systems: A survey, taxonomy, and open issues," *Knowledge-Based Systems*, vol. 189, p. 105124, 2020.

[2] K. Scarfone and P. Mell, "Guide to intrusion detection and prevention systems (idps)," National Institute of Standards and Technology, Tech. Rep., 2012.

[3] D. Mudzingwa and R. Agrawal, "A study of methodologies used in intrusion detection and prevention systems (idps)," in *2012 Proceedings of IEEE Southeastcon*. IEEE, 2012, pp. 1–6.

[4] S. A. R. Shah and B. Issac, "Performance comparison of intrusion detection systems and application of machine learning to snort system," *Future Generation Computer Systems*, vol. 80, pp. 157–170, 2018.

[5] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges," *Cybersecurity*, vol. 2, no. 1, p. 20, 2019.

[6] 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*. IEEE, 2009, pp. 1–6.

[7] A. Sperotto, R. Sadre, F. Van Vliet, and A. Pras, "A labeled data set for flow-based intrusion detection," in *International Workshop on IP Operations and Management*. Springer, 2009, pp. 39–50.

[8] J. Song, H. Takakura, Y. Okabe, M. Eto, D. Inoue, and K. Nakao, "Statistical analysis of honeypot data and building of kyoto 2006+ dataset for nids evaluation," in *Proceedings of the first workshop on building analysis datasets and gathering experience returns for security*, 2011, pp. 29–36.

[9] S. Garcia, M. Grill, J. Stiborek, and A. Zunino, "An empirical comparison of botnet detection methods," *computers & security*, vol. 45, pp. 100–123, 2014.

[10] 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)*. IEEE, 2015, pp. 1–6.

[11] M. Ring, S. Wunderlich, D. Grüdl, D. Landes, and A. Hotho, "Flow-based benchmark data sets for intrusion detection," in *Proceedings of the 16th European conference on cyber warfare and security*, 2017, pp. 361–369.

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

[13] X. Li, F. Bian, M. Crovella, C. Diot, R. Govindan, G. Iannaccone, and A. Lakhina, "Detection and identification of network anomalies using sketch subspaces," in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, 2006, pp. 147–152.

[14] Y. Wang, S. Parthasarathy, and S. Tatikonda, "Locality sensitive outlier detection: A ranking driven approach," in *2011 IEEE 27th International Conference on Data Engineering*. IEEE, 2011, pp. 410–421.

[15] J. Camacho, A. Pérez-Villegas, P. García-Teodoro, and G. Maciá-Fernández, "Pca-based multivariate statistical network monitoring for anomaly detection," *Computers & Security*, vol. 59, pp. 118–137, 2016.

[16] D. Bekerman, B. Shapira, L. Rokach, and A. Bar, "Unknown malware detection using network traffic classification," in *2015 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2015, pp. 134–142.

[17] E. K. Viegas, A. O. Santin, and L. S. Oliveira, "Toward a reliable anomaly-based intrusion detection in real-world environments," *Computer Networks*, vol. 127, pp. 200–216, 2017.

[18] R. Vinayakumar, M. Alazab, K. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman, "Deep learning approach for intelligent intrusion detection system," *IEEE Access*, vol. 7, pp. 41 525–41 550, 2019.

[19] R. Fontugne, J. Mazel, and K. Fukuda, "Hashdoop: A mapreduce framework for network anomaly detection," in *2014 IEEE conference on computer communications workshops (INFOCOM WKSHPS)*. IEEE, 2014, pp. 494–499.

[20] G. Yuan, B. Li, Y. Yao, and S. Zhang, "A deep learning enabled subspace spectral ensemble clustering approach for web anomaly detection," in *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017, pp. 3896–3903.

[21] O. Y. Al-Jarrah, Y. Al-Hammdi, P. D. Yoo, S. Muhaidat, and M. Al-Qutayri, "Semi-supervised multi-layered clustering model for intrusion detection," *Digital Communications and Networks*, vol. 4, no. 4, pp. 277–286, 2018.

[22] G. Osada, K. Omote, and T. Nishide, "Network intrusion detection based on semi-supervised variational auto-encoder," in *European Symposium on Research in Computer Security*. Springer, 2017, pp. 344–361.

[23] Q. P. Nguyen, K. W. Lim, D. M. Divakaran, K. H. Low, and M. C. Chan, "Gee: A gradient-based explainable variational autoencoder for network anomaly detection," in *2019 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2019, pp. 91–99.

[24] "Intelligent security operations." [Online]. Available: https://northwave-security.com/

[25] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[26] N. Marir, H. Wang, G. Feng, B. Li, and M. Jia, "Distributed abnormal behavior detection approach based on deep belief network and ensemble svm using spark," *IEEE Access*, vol. 6, pp. 59 657–59 671, 2018.

[27] B. Zhang, Y. Yu, and J. Li, "Network intrusion detection based on stacked sparse autoencoder and binary tree ensemble method," in *2018 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, 2018, pp. 1–6.

[28] Z. Cheng, C. Zou, and J. Dong, "Outlier detection using isolation forest and local outlier factor," in *Proceedings of the conference on research in adaptive and convergent systems*, 2019, pp. 161–168.

[29] "Google scholar," https://scholar.google.com/intl/nl/scholar/about.html, accessed: 2020-11-11.

[30] A. Martín-Martín, E. Orduna-Malea, M. Thelwall, and E. D. López-Cózar, "Google scholar, web of science, and scopus: A systematic comparison of citations in 252 subject categories," *Journal of Informetrics*, vol. 12, no. 4, pp. 1160–1177, 2018.

[31] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, and A. Hotho, "A survey of network-based intrusion detection data sets," *Computers & Security*, vol. 86, pp. 147–167, 2019.

[32] D. S. Terzi, R. Terzi, and S. Sagiroglu, "Big data analytics for network anomaly detection from netflow data," in *2017 International Conference on Computer Science and Engineering (UBMK)*. IEEE, 2017, pp. 592–597.

[33] H. Liu and B. Lang, "Machine learning and deep learning methods for intrusion detection systems: A survey," *Applied Sciences*, vol. 9, no. 20, p. 4396, 2019.

[34] I. Ahmad, M. Basheri, M. J. Iqbal, and A. Rahim, "Performance comparison of support vector machine, random forest, and extreme learning machine for intrusion detection," *IEEE access*, vol. 6, pp. 33 789–33 795, 2018.

[35] S. Potluri, S. Ahmed, and C. Diedrich, "Convolutional neural networks for multi-class intrusion detection system," in *International Conference on Mining Intelligence and Knowledge Exploration*. Springer, 2018, pp. 225–238.

[36] K. A. Costa, L. A. Pereira, R. Y. Nakamura, C. R. Pereira, J. P. Papa, and A. X. Falcão, "A nature-inspired approach to speed up optimum-path forest clustering and its application to intrusion detection in computer networks," *Information Sciences*, vol. 294, pp. 95–108, 2015.

[37] M. Ahmed and A. N. Mahmood, "Novel approach for network traffic pattern analysis using clustering-based collective anomaly detection," *Annals of Data Science*, vol. 2, no. 1, pp. 111–130, 2015.

[38] C. Nixon, M. Sedky, and M. Hassan, "Autoencoders: A low cost anomaly detection method for computer network data streams," in *Proceedings of the 2020 4th International Conference on Cloud and Big Data Computing*, 2020, pp. 58–62.

[39] A. Verma and V. Ranga, "Machine learning based intrusion detection systems for iot applications," *Wireless Personal Communications*, vol. 111, no. 4, pp. 2287–2310, 2020.

[40] A. Ahmim, L. Maglaras, M. A. Ferrag, M. Derdour, and H. Janicke, "A novel hierarchical intrusion detection system based on decision tree and rules-based models," in *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. IEEE, 2019, pp. 228–233.

[41] N. Koroniotis, N. Moustafa, E. Sitnikova, and B. Turnbull, "Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset," *Future Generation Computer Systems*, vol. 100, pp. 779–796, 2019.

[42] A. Khraisat, I. Gondal, P. Vamplew, J. Kamruzzaman, and A. Alazab, "A novel ensemble of hybrid intrusion detection system for detecting internet of things attacks," *Electronics*, vol. 8, no. 11, p. 1210, 2019.

[43] G. Kim, S. Lee, and S. Kim, "A novel hybrid intrusion detection method integrating anomaly detection with misuse detection," *Expert Systems with Applications*, vol. 41, no. 4, pp. 1690–1700, 2014.

[44] M. M. Rathore, A. Ahmad, and A. Paul, "Real time intrusion detection system for ultra-high-speed big data environments," *The Journal of Supercomputing*, vol. 72, no. 9, pp. 3489–3510, 2016.

[45] K. R. Vigneswaran, R. Vinayakumar, K. Soman, and P. Poornachandran, "Evaluating shallow and deep neural networks for network intrusion detection systems in cyber security," in *2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. IEEE, 2018, pp. 1–6.

[46] M. E. Aminanto, R. Choi, H. C. Tanuwidjaja, P. D. Yoo, and K. Kim, "Deep abstraction and weighted feature selection for wi-fi impersonation detection," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 3, pp. 621–636, 2017.

[47] G. Maciá-Fernández, J. Camacho, R. Magán-Carrión, P. García-Teodoro, and R. Therón, "Ugr '16: A new dataset for the evaluation of cyclostationarity-based network idss," *Computers & Security*, vol. 73, pp. 411–424, 2018.

[48] W. Anani and J. Samarabandu, "Comparison of recurrent neural network algorithms for intrusion detection based on predicting packet sequences," in *2018 IEEE Canadian Conference on Electrical & Computer Engineering (CCECE)*. IEEE, 2018, pp. 1–4.

[49] P. Casas, J. Mazel, and P. Owezarski, "Unsupervised network intrusion detection systems: Detecting the unknown without knowledge," *Computer Communications*, vol. 35, no. 7, pp. 772–783, 2012.

[50] M. H. Bhuyan, D. Bhattacharyya, and J. K. Kalita, "A multi-step outlier-based anomaly detection approach to network-wide traffic," *Information Sciences*, vol. 348, pp. 243–271, 2016.

[51] G. Xie, K. Xie, J. Huang, X. Wang, Y. Chen, and J. Wen, "Fast low-rank matrix approximation with locality sensitive hashing for quick anomaly detection," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[52] X. Zhang, W. Dou, Q. He, R. Zhou, C. Leckie, R. Kotagiri, and Z. Salcic, "Lshiforest: a generic framework for fast tree isolation based ensemble anomaly analysis," in *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE, 2017, pp. 983–994.

[53] M. Goldstein and A. Dengel, "Histogram-based outlier score (hbos): A fast unsupervised anomaly detection algorithm," *KI-2012: Poster and Demo Track*, pp. 59–63, 2012.

[54] F. Salo, A. B. Nassif, and A. Essex, "Dimensionality reduction with ig-pca and ensemble classifier for network intrusion detection," *Computer Networks*, vol. 148, pp. 164–175, 2019.

[55] C. Callegari, S. Giordano, and M. Pagano, "An information-theoretic method for the detection of anomalies in network traffic," *Computers & Security*, vol. 70, pp. 351–365, 2017.

[56] J. Zhang, H. Li, Q. Gao, H. Wang, and Y. Luo, "Detecting anomalies from big network traffic data using an adaptive detection approach," *Information Sciences*, vol. 318, pp. 91–110, 2015.

[57] B. E. Strom, A. Applebaum, D. P. Miller, K. C. Nickels, A. G. Pennington, and C. B. Thomas, "Mitre att&ck: Design and philosophy."

[58] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[59] "Keras," https://keras.io/, accessed: 2021-04-08.

[60] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[61] R. A. Peterson and J. E. Cavanaugh, "Ordered quantile normalization: a semiparametric transformation built for the cross-validation era," *Journal of Applied Statistics*, 2019.

[62] "Scikit-learn quantiletransformer," https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.QuantileTransformer.html#sklearn.preprocessing.QuantileTransformer, accessed: 2021-06-09.

[63] A. Thomas, V. Feuillard, A. Gramfort, and S. Clémençon, "Learning hyperparameters for unsupervised anomaly detection." in *ICML, Anomaly Detection Workshop*, 2016.

[64] J. Yang, S. Rahardja, and P. Fränti, "Outlier detection: how to threshold outlier scores?" in *Proceedings of the international conference on artificial intelligence, information processing and cloud computing*, 2019, pp. 1–6.

[65] C. Krügel, T. Toth, and E. Kirda, "Service specific anomaly detection for network intrusion detection," in *Proceedings of the 2002 ACM symposium on Applied computing*, 2002, pp. 201–208.

[66] A. Sperotto and A. Pras, "Flow-based intrusion detection," in *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*. IEEE, 2011, pp. 958–963.

[67] M. Labonne, A. Olivereau, B. Polvé, and D. Zeghlache, "Unsupervised protocol-based intrusion detection for real-world networks," in *2020 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2020, pp. 299–303.

[68] "Scikit-learn cheatsheet," https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html, accessed: 2021-06-08.

[69] "Zeek," https://docs.zeek.org/en/current/index.html, accessed: 2021-02-04.

[70] "Scikit-learn labelencoder," https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html, accessed: 2021-03-23.

[71] "Scikit-learn mutualinformation," https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.mutual_info_classif.html, accessed: 2021-04-08.

[72] "Zeek ttl script," https://github.com/JKeijer/Zeek_scripts, accessed: 2021-04-08.

[73] D. Pérez, S. Alonso, A. Morán, M. A. Prada, J. J. Fuertes, and M. Domínguez, "Comparison of network intrusion detection performance using feature representation," in *International Conference on Engineering Applications of Neural Networks*. Springer, 2019, pp. 463–475.

[74] G. C. Fernández and S. Xu, "A case study on using deep learning for network intrusion detection," in *MILCOM 2019-2019 IEEE Military Communications Conference (MILCOM)*. IEEE, 2019, pp. 1–6.

[75] T. Truong-Huu, N. Dheenadhayalan, P. Pratim Kundu, V. Ramnath, J. Liao, S. G. Teo, and S. Praveen Kadiyala, "An empirical study on unsupervised network anomaly detection using generative adversarial networks," in *Proceedings of the 1st ACM Workshop on Security and Privacy on Artificial Intelligence*, 2020, pp. 20–29.

[76] H. He, X. Sun, H. He, G. Zhao, L. He, and J. Ren, "A novel multimodal-sequential approach based on multi-view features for network intrusion detection," *IEEE Access*, vol. 7, pp. 183 207–183 221, 2019.