# Good || Evil: Defending Infrastructure at Scale with Anomaly and Classification Based Network Intrusion Detection

August, 2021


Philippe Partarrieu*
M.Sc. Security and Network Engineering
University of Amsterdam
philippe.partarrieu@os3.nl

Philipp Mieden*
M.Sc. Security and Network Engineering
University of Amsterdam
philipp.mieden@os3.nl

Giovanni Sileno†
g.sileno@uva.nl

Joao Novais Marques†
joao.novaismarques@kpn.com

Jordi Scharloo†
jordi.scharloo@kpn.com

* student † supervisor



***Abstract*— We develop and evaluate a pipeline for extracting and analysing observations from network traffic, using state of the art machine learning and deep learning algorithms to recognise malicious patterns in network communication, without the need for static detection rules. The evaluation is conducted on the CIC-IDS-2018 dataset, a modern and large scale scenario that includes multiple common attack classes inside a multi-department corporate network infrastructure hosted on AWS. Our experiments seek to evaluate the detection performance of various algorithms, as well as the impact of different model parameters, to determine which delivers the highest value for security threat analysts. For this purpose, we compare the results of binary classification, to more fine grained multi-class attack classification, and investigate the possibility of knowledge transfer between network topologies as well as limitations for real-world deployment of anomaly- and classification-based network intrusion detection systems.**



***Index Terms*—network security, machine learning, artificial intelligence, data science, intrusion detection, deep learning**


## I. Introduction

Computer networks are frequently attacked with new or previously unseen malware in order to steal information or cause financial damage. Defending effectively against these has become a difficult task because the current signature based detection mechanism only detects known malicious programs or behaviours [1]. Even if a signature exists for a known strain of malware, obfuscation techniques can be used to avoid detection, either by making changes to the code base, or by updating key attack characteristics and used infrastructure. In addition to this problem, the signature databases are growing, and need to be kept up-to-date in order to protect against latest threats.

Anomaly detection attempts to solve the problem of detecting previously unknown attack variations, by modelling the normal behaviour of clients on a computer network and alerting an analyst of any deviations from these patterns. Machine learning has been an active area of research over the past 20 years, and is known to work well for many similar data science problems. As a result, nowadays there are many different algorithms and frameworks to choose from.

Machine Learning (ML) algorithms range from classical fingerprinting and frequency based observations, over decision tree based methods, to Deep Learning algorithms and the use of Deep Neural Network (DNN), that mimic the behaviour of the human brain in order to make a classification based on learned attributes and data characteristics.

A core challenge for the application of anomaly based detection methods, is the selection, collection and extraction of feature vectors that are fed into the classification algorithms. This process not only has to be secure, as it is parsing untrusted and potentially malicious input, but also efficient in terms of computational resources, in order to satisfy real time analysis requirements and handle the large volume of traffic in modern networks.

A distinction has to be made between two different methodologies for training a ML classification algorithm. In supervised learning, an algorithm is fed input-output pairs and learns a function that maps the input to the output. In unsupervised learning the algorithm learns patterns in the input even though no explicit feedback is supplied. The latter is more generic and less specific to the target environment, whereas the supervised approach can offer a higher accuracy and fewer false positives [2].

## II. Background & Related Work

### A. Network Intrusion Detection Systems

Anomaly-based Intrusion Detection Systems, especially unsupervised learning algorithms, have been shown to be highly

capable at detecting previously unknown attacks such as zero-days, obfuscated attacks or variations of known malicious patterns [3].

In 2016, Veeramachaneni et al. designed an anomaly detection system that employs a security expert to classify the alerts raised by an outlier detection model. The classified data is then fed into a supervised learning algorithm. This combination of supervised and unsupervised learning with an analyst in the loop allows the system to detect more anomalies with a higher sensitivity but the training takes time and human resources. The study shows that the system may take up to 3 months to outperform a simple outlier detection model. We argue that such long convergence time could be a problem for the requirements of modern network infrastructure, due to its fast paced change over time.

### *B. Outlier Detection Algorithms*

There are many anomaly detection algorithms e.g. K-Nearest Neighbours, Support Vector Machine, Isolation Forests. The applicability of each will vary between domains and we will only discuss research that investigates outlier detection algorithms specifically for network intrusion detection.

In 2003, Lazarevic et al. [4] found that the Local Outlier Factor (LOF) was the most promising at detecting network intrusions compared to a Neural Network (NN), a Mahalanobis, and an Support Vector Machine (SVM) based approach. In 2013, Emmott et al. [5] performed a comparative study and found that Isolation Forests, an algorithm discovered in 2008 [6], was the top performer at detecting anomalies toe-to-toe with Ensemble Gaussian Mixture Model (EGMM), LOF and SVM.

Isolation Forest is a very interesting choice since it is the best performing model in terms of resources: it has a linear time complexity and low memory requirements. However Isolation Forests suffers from anomaly masking, which refers to the problem that too many anomalies are concealing their own presence. Indeed the algorithm relies on the intuition that anomalies are susceptible to isolation under random partitioning but this assumption fails when anomaly clusters are large and dense [6].

This raises an interesting insight in network intrusion detection and for traffic analysis in general: malicious behaviour isn't necessarily anomalous e.g. Denial of Service (DoS) attacks against intentionally exposed services, where a public resource is requested frequently enough to exhaust the capacity of the underlying infrastructure. Furthermore anomalies aren't necessarily malicious, but might still be worth reporting for the purpose of defect detection e.g. system malfunctions due to misconfigurations.

### *C. Deep Learning*

Although Deep Neural Networks have been making significant improvements in recent years [7] [8] and can score high results for classification tasks, the black-box nature of these algorithms makes it hard to justify why an anomaly has been flagged as such [9]. Deep learning models show great performance as long as the data seen during tests is similar to the training data. Effects of substantially different patterns on the prediction accuracy, are hard to estimate and require manual testing. This poses a problem for the application of deep learning to network security monitoring.

It has been shown that a mismatch in distribution will lead deep learning models to give high confidence predictions on anomalies [10], meaning that the algorithm is easily fooled when the ratio of benign to malicious events changes substantially. In 2019, Hendrycks et al. [8] try to address this problem in the context of natural language processing and vision tasks, by exposing a deep learning model to outliers. They conclude that outlier exposure significantly improved the detection performance of the tested classifiers.

Literature by Mukkamala et al. [11] has shown that an ensemble approach, combining various algorithms (NN, SVM, and Multivariate Adaptive Regression Splines) is superior in terms of classification accuracy over the individual approach. The intuition behind these results stems from the fact that malicious network behaviour may exhibit different statistical properties depending on the attack class. Each model has individual strengths and weaknesses and may be better suited at detecting a given attack class but performs worse at another. The ensemble approach shows that these models can work synergistically.

In 2018, Mirsky et al. [12] showed that Kitsune, an ensemble of auto-encoders, performed better than to 2 individual offline algorithms, Isolation Forest and Gaussian Mixture Model (GMM), and 2 individual online algorithms, pcStream and incremental GMM.

### *D. Datasets*

Arguably one of the most important variables when it comes to training Machine Learning algorithms is the dataset. Academic research in anomaly detection is known to suffer from a lack of realistic and publicly available datasets [5]. Indeed, synthetic datasets don't exhibit real-world validity and therefore it becomes hard to decide whether the algorithm trained on such data works well in real-world settings. Many real-world datasets are kept private due to privacy concerns and the few public ones are often heavily redacted as stated by Sharafaldin et al. [13].

Emmott et al. [5] propose a systematic benchmarking methodology for real data datasets pertaining to anomaly detection by measuring the data according to three dimensions: point difficulty, relative frequency of anomalies, and clusteredness. They are able to show that these properties are crucial in influencing the behaviour of anomaly detection algorithms.

## III. Problem and Research Gap

Signatures are often using information that can easily be changed by the authors of the malicious code. For example, many detection rules for the signature based intrusion detection systems *Snort* and *Suricata*, rely on specific domain names or IP addresses of computer infrastructure that is

known to be used by criminals. However, these identifiers can be easily changed or the command and control server can be moved to a different hosting provider, and the rule is no longer effective in detecting the threat. Detection on the endpoints itself faces a similar problem, as the use of certain bit patterns or symbol names in binary files is not effective at preventing attackers with access to the source code from making modifications that will evade detection, by using binary obfuscation techniques.

As complete security of endpoint systems cannot be guaranteed in practice, due to the aforementioned reasons, network security operations must take infections and incidents into account and develop mitigations and detection mechanisms. A key factor for the successful contention of malicious activity is time: a case study from Mandiant published in 2012 showed that the median time from the start of an intrusion to the incident response is over 240 days [14]. Damage can potentially be prevented thanks to timely detection and response by an automated system. Flow collection for high throughput networks is often sampled, otherwise the hardware would not support the capture process [15]. This potentially introduces a gap in visibility, as attack packets might be missed. Some attacks might use techniques that are not immediately obvious to a human analyst, for example fragmentation attacks, but cause patterns in the data points that an algorithm can detect. The implementation of this however, is a resource intensive task that requires stream reassembly, and can lead to denial of service attacks if implemented incorrectly [16].

An effective way to get alerted about suspicious activity on endpoints is to analyse the network traffic generated by those hosts, as malware in a majority of cases contacts a command and control server, to receive further commands or upload gathered data or encryption keys, in order to provide value for an attacker. There are exceptions to this that depend on the nature of the attack: consider malware that only aims to wipe the endpoints and infection via USB drive. In that case, no network communication is required for success of the mission, and as a consequence, detecting such attacks is out of scope for this project. We focus on attacks that leave traces on the network exclusively.

Detecting network attacks comes with the challenge of detecting rare events. This poses a problem for commonly used algorithms in the data science world, as those expect a similar distribution of the classes used to describe the data. To deal with this limitation, several approaches are available. Manual investigation of network traffic pattern is extremely time intensive, automated solutions that apply preselection for the events the analyst will be presented, can potentially drastically reduce the workload imposed on the human operator.

The choice of learning strategy comes with limitations for the operational phase, and for each algorithm, there are many different configurations and possible optimisations. Which algorithm, configuration and feature selection works best for the detection of attacks in computer networks, is still an active area of research with no ultimate solution yet in sight. Furthermore, academic experiments on machine learning based network intrusion detection often make use of expensive supercomputers, or cloud services with high performance configurations, that are out of reach for other researchers. To address this, our experiments are conducted entirely on commodity hardware. Another problem with the current state of research in the area is reproducibility: many publications do not include instructions or code to allow for independent verification of the presented results within reasonable time. Experiments are often restricted to a single domain of algorithms, whereas a comparison within those might provide additional value. Lastly, some work in the field still makes use of outdated datasets, for example the NSL-KDD dataset [17], which is based on the over 20 years old KDD-99 dataset, and as a consequence no longer representative for real-world traffic. We therefore seek to evaluate the state of the art algorithms and compare their performance on a modern dataset, while creating an open source test bed that can be used in future experiments and for the reproduction of our results.

## IV. Research Question

We are interested in evaluating state of the art open source solutions to classify malicious behaviour in computer networks on a modern large scale dataset.

During this research project, we intend to answer the following questions:

1) Which features prove the most useful for the detection task?
2) What performance can be achieved with the proposed implementation with regard to throughput and hardware requirements?
3) Which algorithms are best suited for the task of anomaly detection in computer networks?

## V. Methodology

### A. Dataset

The *CIC-IDS-2018* dataset was published by Sharafaldin et al. [13] and to our knowledge is the most extensive publicly available network intrusion detection dataset to date. It includes a variety of attacks, ranging from denial of service and distributed variant Distributed Denial of Service (DDoS) to infiltration, injection and bruteforce attacks. It comes as a set of raw packet captures and provides uni-directional network flow data with many statistical features, and with attack labels, intended to be used by ML algorithms. Recent attack methodologies are carried out against a five department corporate infrastructure hosted on Amazon Web Services (AWS), that is connected via a network infrastructure provided by 30 servers, including in-house mail, file server and Windows active directory services. A large scale attack operation is simulated from *50* dedicated machines to conduct attacks against the target services and endpoints.

### B. Dataset Limitations

This section lists issues we encountered with the dataset and makes suggestions for possible improvements.

*1) Incomplete Network Flow Data*

Provided network flow records where missing address information, namely the FlowID, SrcIP, DstIP and the SrcPort. After contacting the dataset authors we received the full enriched network flow dataset.

*2) Non Reproducible Labelling*

The labelling tool or strategy used has not been open sourced by the CIC-IDS-2018 dataset authors, and as such cannot be reproduced. Furthermore we noticed that the paper states that the port information during attacks was used for the labelling effort, but no port information is provided in the table that lists the attacks.

*3) Incomplete Packet Captures*

Packet captures for Thursday-15-02-2018 were missing attack traffic and seem to have been cut short, as the traffic ends at 9am, as a consequence no attack labels are present in the connection audit records for this day. However the provided network flow files do contain attacks on this day and could be used in our experiments. The affected day is marked in the result tables.

*4) Absence of Normal Days*

No days that contain only normal traffic have been provided. This would have been useful for validation purposes to measure the amount of false predictions during days with exclusively benign activity.

*5) Dataset Imbalance*

The *CIC-IDS-2018* dataset contains an imbalanced ratio of attack traffic to normal traffic, which impacts the choice of metrics for evaluation. This ratio varies from 5-20% for the provided network flow data, and between 1-5% for the connection audit records extracted by our tooling, as these aggregate the flows in both directions and ignore sub-flows triggered via RST or FIN flags.

Behaviour of anomaly detection algorithms varies depending on the ratio of anomalies. If anomalies are rare, algorithms that fit a model to "normal" samples may do well. Inversely if anomalies are common, algorithms that fit a model to malicious samples may do well.

Different strategies exist to deal with dataset imbalance:

- Class weights are calculated based on the ratio of the positive to the negative class, and force the algorithm to pay more attention to underrepresented classes. Each class receives a value for its weight.
- Oversampling consists in selectively picking samples of underrepresented classes and duplicating them, in order to increase their share in the dataset.
- Outlier exposure as described in [8], training a deep learning model against a dataset of outliers leads to better detection of unseen anomalies.

### C. Dataset Exploration

Each of the six different types of attacks in the dataset has different characteristics. The following sections will shortly describe the expected characteristics for each of those attacks and explore advanced variations for some attacks that could be used by an attacker to complicate detection, or evade static rules.

*1) Botnet Activity*

Infected workstations send periodic beacons or exfiltrated data to the command and control server of the attacker, in this case screenshots from the victim machines are continuously uploaded to a service from the attacker. This creates a static communication pattern from the inside network to the outside world, possibly also with a static data transfer size. In the CIC dataset, a fixed dimension screenshot is repeatedly transferred to a command and control server by the infected machines. Advanced implementations of such an attack might introduce random jitter to make detection harder, or vary the payload size with each transfer.

*2) Infiltration*

User machines start to behave abnormally after initial infection. Abnormal activity might include network scanning for reconnaissance, data exfiltration or download of further malware. This can happen potentially from a legitimate service like for example Dropbox and over an encrypted TLS connection. Additionally an attacker can expand his presence in the compromised network using lateral movement. After successful infiltration, data exfiltration can potentially happen very slowly or tunnelled in another protocol like Internet Control Message Protocol (ICMP) or Domain Name Service (DNS), besides the use of an encrypted Transport Layer Security (TLS) connection over an allowed port like 443, for Secure Hypertext Transfer Protocol (HTTPS) traffic. An interesting indicator are the connections from the attacker machine that are used to control the victim machine during further malicious activity, in this case using a meterpreter reverse shell, over an unencrypted Transport Control Protocol (TCP) connection. Network scanning creates a similar static pattern of a host that contacts another host on many different ports, potentially over a longer period of time, to avoid detection. Simple scanning configurations might probe ports in sequential order, while configurations that try to be more stealthy could randomise this order.

*3) Bruteforce*

Multiple attempts to access a resource in a short period of time, from a single or multiple hosts. A small amount of data is sent, as authentication data like credentials are usually very small, or in case of using certificates for authentication has a fixed length. In case the features include the status code for authentication operations, a series of authentication error codes can be observed. Several open source tools exist for launching dictionary attacks against exposed services that support authentication [18]. We expect that it can pose a challenge for an algorithm to determine whether it is a program trying to brute force access to an account, or the legitimate user that made a typo when entering his password, especially

when only a low number of attempts are used, in combination with randomised delay between authentication attempts.

*4) Denial of Service*

This attack type seeks to disrupt the availability of a system, either by generating a high frequency of incoming requests, from a single, or many different hosts in case of the distributed variant, or by crashing the system entirely. A pattern that can be observed is that a certain resource, for example a public facing service, is more frequently requested than usual. This however might not always be malicious, consider an online shop that launches a sale, and as a consequence expects a high amount of visitors over a short time frame. In some cases, even a single packet can be enough to crash an entire application and render it unusable for its users [19]. Due to the statistical patterns, especially those in resource access frequency, we consider these attacks to be detected with a rather high detection rate.

*5) Injection*

Injection attacks often allow for the execution of malicious code, and some can be used to gain remote control over a target [19]. The dataset contains examples of SQL injections, which lead to the execution of attacker controlled queries on a SQL database backend, enabling data theft or destruction. The SQL backend communicates over a TCP connection, and as such the traffic and exchanged amount of data can be observed. The database connection might be kept open and reused for multiple queries.

### D. Feature Extraction

In order to determine which features are the most valuable for the task of network intrusion detection, we extended the connection audit records provided by the Netcap framework [20]. A key goal for the feature extractor is performance, as it is crucial for the real-world usage scenario and when working with large amounts of packet traces. In addition extensibility is very important, in order to allow for quick adjustments for different usage scenarios and experiments with new features. Our tooling is completely open source to allow the reproduction of our results and enable future research in the area of network intrusion detection. For the experiments, Netcap version *v0.6.0* has been used.

To identify features of interest and encode them correctly, we studied common techniques for feature engineering as outlined by Zheng et al. [21] and applied them to our scenario. Furthermore techniques for exploratory data analysis as described by Collins [22] have been used to identify features of interest.

The feature extractor is configurable and allows the features to be generated either as raw values, or encoded as numerical values for an ML algorithm to use. We implemented support for structured labelling of all provided audit record types in the Netcap core, so that the extracted data points can be used for supervised training algorithms, that require ground truth during the training phase.

A feature is a numeric representation of a raw data point, and derives from the type of data that is available. The number and quality of available features greatly impacts the performance of a model: if there are not sufficient informative features, the model will be unable to perform the predictive task [21]. Feature Extraction should be parallelised to take advantage of multi-core processors. Concurrency causes problems though, such as race conditions due to shared state among workers. Ideally, observations from network communication should be lightweight to calculate, but still expressive enough to capture general network trends and the behaviour of individual hosts. Our solution made use of bi-directional network flow summaries, referred to as connection audit records in our experiments. This requires keeping only minimal state, and aggregates subflows, which compresses the amount of generated events.

We chose a deliberately simple set of features for establishing a baseline. Additional features can always be added later and their effect on prediction quality measured, as the number of features and their complexity greatly affects the performance of their collection. As a consequence, their contribution to the detection task must be carefully evaluated for each feature and different feature combinations. Included information are timestamps, the total connection duration, address information from the link and network layer, as well as data transfer statistics, such as the number of bytes transferred in each direction. In addition, we added counters for the number of TCP flags seen, as well as tracking of the mean TCP window size for each connection. For the entire dataset, we yield *39,850,471* connection records, that have an uncompressed size of *7.1G* in Comma Separated Values (CSV) format. When compressed using zip, the data size is reduced by *74%* to *1.9G* on disk.

### E. Choice of Metrics

Accuracy is not a suitable metric when dealing with imbalanced data, since the data imbalance will allow achieving high scores without making any useful predictions. In order to compensate for this and take the relevance of the different error types into account, we use the F1 score to evaluate the performance of the algorithms in our experiments. The F1 score is the harmonic mean of recall and precision. The precision metric measures how many selected elements are relevant, whereas the recall metric measures how many relevant items have been selected. The different error types are shown in figure 1.

A true positive is the correct classification of an anomalous sample, while a true negative is the correct classification of a benign sample. A false positive refers to benign network traffic that is incorrectly classified as an attack. Lastly, a false negative is an attack that was not detected correctly by the algorithm and was instead labelled benign.

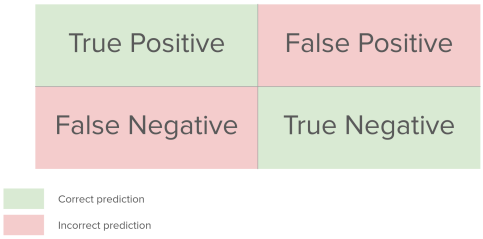


Fig. 1: Confusion matrix

The recall is calculated by dividing the amount of true positives by the amount of true positives and false negatives:

$$recall = \frac{true\ positives}{true\ positives + false\ negatives}$$

The precision is calculated by dividing the amount of true positives by the amount of true positives and false positives:

$$precision = \frac{true\ positives}{true\ positives + false\ positives}$$

The harmonic mean is a well suited metric for analysing imbalanced data sets, since it effectively penalises the algorithm when either the precision or recall value is low [23].

$$F1\ score = 2 * \frac{recall * precision}{recall + precision}$$

For monitoring the performance of a neural network, a loss function is used, which takes multiple metrics of the model during operation, and produces a numeric value that describes the performance of the model. The closer the result of the loss function is to zero, the better the model is performing [24]. In order to avoid overfitting the model to the training dataset, we use early stopping to stop the training phase once the model no longer makes any notable improvements. To achieve this, the delta between the two most recent loss values is checked after each epoch. Once this delta drops below the commonly used threshold for successful learning of *0.001*, the training phase is stopped, and the model that had the lowest loss at the end of an epoch is selected from all trained versions of the model.

### F. Deep Neural Network Class Weight Calculation

To calculate the weights in order to adjust to the class imbalance, the following recommended formula is used:

$$(1/samples\_class) * (total/2.0)$$

Scaling the result by *total / 2* is done to keep the loss at a similar magnitude, while the sum of the weights of all examples stays the same, as recommended by the Tensorflow guide for classification on imbalanced datasets [25].

In our experiments, we observed that the class weights calculated with this formula did not always to perform well, and therefore started to experiment with different variations of the class weight calculation. We found the most effective way to influence the behaviour of the model towards the positive class to manipulate it using a factor, the result for the negative class is used unchanged. This way the focus on the positive class can be tweaked, after creating a starting point using the previous formula. Through manual experimentation with binary classification, we determined the best working factor to be a rather small one, that is close to zero, effectively making the model much less sensitive to the positive class. Our analysis has shown that models trained using the recommended formula suffered from over sensitivity towards the positive class. By using a factor, we can either up- or downscale the final value calculated for the positive class, by using a factor above 1, or between 0 and 1.

$$(1/positive\_class) * (total/2.0) * pos\_factor$$

We observed the value of the share of the positive class in percent to deliver best results when used as a factor, our tests with tweaked class weights make use of this optimisation and compare it against the effect of using the standard formula. As an example, given a *1.7%* share of the positive class in the dataset, the factor used for the tweaked class weight calculated would be *0.017*, drastically reducing the weight for the positive class.

### G. Experiment Design

The experiments all operate on the *CIC-IDS-2018* dataset and can be divided into deep learning and machine learning algorithms. Experiments using the DNN and Kitsune autoencoder ensemble operate on the connection audit records produced by the Netcap framework. The machine learning experiments operate on the provided *CIC-IDS-2018* flow data, that contains over 80 statistical features for uni directional network flows. All experiment code and logs are publicly available to reproduce and independently verify our results [26]. All used features, parameters and configuration values are listed in appendix.

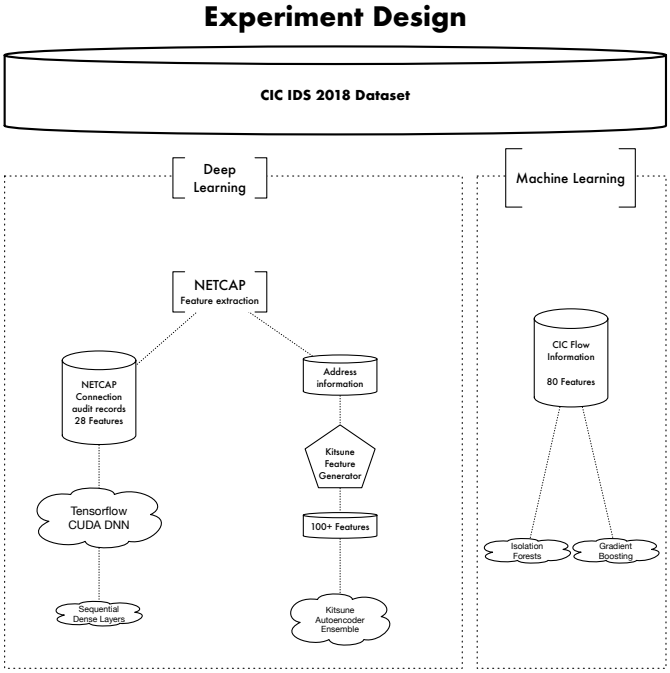


Fig. 2: Experiment design overview

## VI. Implementation

### A. Labelling

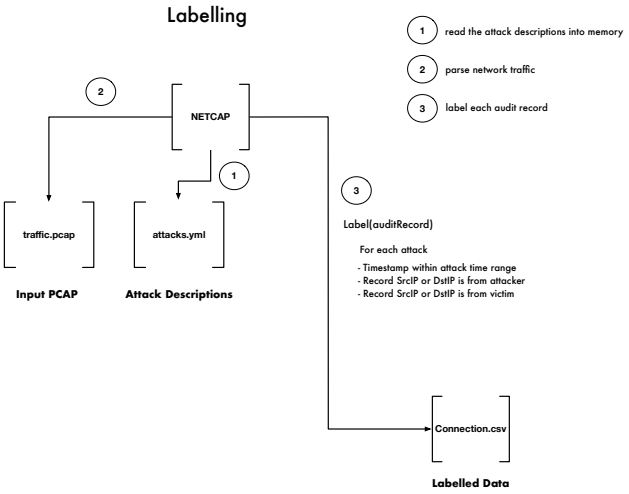


Fig. 3: Labelling

In order to provide label information for the use of supervised machine learning, attaching labels to the audit records produced by the Netcap framework has been implemented. To ensure to ensure the labelling logic works correctly we implemented unit tests, and explicitly included edge cases such as marking events that appear on the edge of an attack time window. Labelling can be applied to any audit record type from Netcap, that implements its central audit record interface. The procedure of loading attack information from a YAML file from disk, and applying it to the generated audit records from the Netcap core for a single day of the dataset is shown in Figure 3. In addition, we generate a plot for visual verification of the assigned labels over the time period of the traffic capture in a configurable interval, as shown in Figure 4.

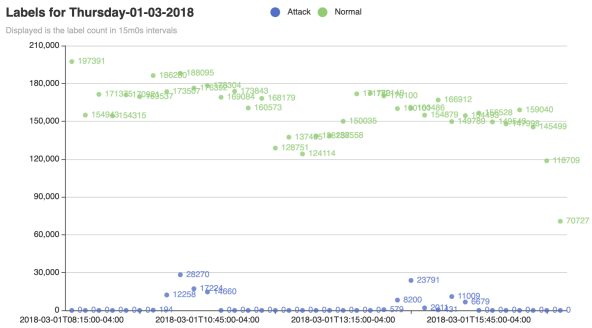


Fig. 4: Thursday Labels

Our labelling logic allows to configure the exact attack matching procedure, in order to avoid marking legitimate victim background traffic as malicious, such as an OS update check or mail traffic, that happens coincidentally during an attack but is not related to it. By default, only traffic between a victim and an attacker IP address within the time frame of an attack will yield a label for the attack. This behaviour can be changed if needed, to also label traffic between the victim machine and other hosts as belonging to the attack, if it happened in the time interval of an attack. We realised this would be needed to mark the malicious activity on the victims behalf after infection in the infiltration scenario, as the victim machine is controlled via a reverse shell by the attacker, and conducts further malicious activity such as network scanning and the download of further malicious software. For scenarios like denial of service or brute force attacks, labelling victim traffic to other hosts than the attacker machines does not make sense and is therefore disabled in our experiments.

### B. Data Encoding

Categorical data like alphanumerical identifiers must be transformed into numeric values, before they can be supplied to an algorithm for analysis. Multiple approaches for encoding categorical data exist, we will briefly discuss two common encoding schemes that we considered for our experiments:

*1) Enumeration*

Each unique categorical value will be assigned a unique numeric value, starting at zero and this index is incremented sequentially for every new value that arrives. This way, the only relation captured with enumeration strategy is the time of first appearance of a value. A downside to this approach is that all unique values must be held in memory for the duration of the system operation. Purging old values might cause unexpected behaviour and impair the efficiency of the detection algorithm.

*2) One Hot Encoding*

A new column is added for every unique value of a feature and set to one for each row of the data that has this value.

This means all possible values of a feature must be known a priori, as the network will lack the required inputs otherwise.

We chose enumeration because it does not alter the feature dimension. For one hot encoding all possible categorical values for the column must be known in advance, so that the inputs for the model can be created and the model trained on their inputs. One hot encoding increases the dimensionality for the dataset dynamically, since every unique value creates a new column. Each DNN model is created for a specific number of features, the so called input shape. In order to work with data points that did not exist during the training phase, the model would need to be retrained, otherwise the new inputs would need to be discarded for the model to work with the new data. For this reason, we determined the enumeration approach to be best suited for our specific use case, especially considering the later deployment of a classification based network intrusion detection system.

### C. Data Normalisation

Numeric values must be normalised to reside within a certain threshold, in order for the DNN to recognise them best. We used the standard normal distribution (*zscore*), for this purpose, because it is known to work well in this domain [21]. An alternative to this is be the min max normalisation strategy, which was also evaluated in our DNN experiments.

## VII. Algorithms

A key consideration in unsupervised ML for network intrusion detection is that anomalies are not drawn from a well-defined probability distribution. Since this is an adversarial setting, attackers can potentially try to defeat the distributional assumptions the model makes.

Many ML algorithms assume that the entire dataset is available for training. In the online paradigm, the data in incrementally fed to the algorithm in a non-blocking way. The downside of this approach is that statistics must be calculated on the fly, whereas offline algorithms can operate full knowledge of the dataset distribution.

| Model | Online | Supervised | Deep Learning |
|---|---|---|---|
| Deep Neural Network | ✗ | ✓ | ✓ |
| Auto Encoders | ✓ | ✗ | ✓ |
| Gradient Boosting | ✗ | ✓ | ✗ |
| Isolation Forest | ✗ | ✗ | ✗ |

Fig. 5: Model Comparison

### A. Deep Neural Networks

For running the DNN experiments we used Tensorflow, an industry standard open source framework for machine learning tasks, that offers support for multiple different computational backends including GPUs, a built-in analysis and computational graph exploration frontend called Tensorboard and support to run in a cluster with multiple nodes [27].

We chose a deliberately small network for the baseline experiments, and experimented with different sizes to measure the impact on detection performance.

The DNN experiments were executed on a *GEFORCE RTX 3090*. Processing six million connection audit records during training and testing took approximately two seconds per epoch with a batch size of 2048 connections and when best model selection is disabled.

The DNN should never be evaluated on data it has already seen in the training phase. We therefore split the data into a training and an evaluation batch, and experimented with different ratios. The baseline experiments use 75% of the dataset for training and 25% for testing. Based on the loss calculated during the training phase, the training process will stop once the development of the loss surpasses a certain threshold. This is done in order to avoid overfitting the model, and prevent yielding a biased model that performs extremely well in the training dataset, but poorly when exposed to data from a different source. When the training process has been early stopped, the model with the lowest loss is selected for the testing phase. A patience value of three epochs is set, to avoid the training process from being stopped prematurely.

### B. Isolation Forest

The intuition behind Isolation Forest relies on the idea that anomalies are more susceptible to isolation under random partitioning. To get the best results, it requires an explicit *contamination rate* which is the ratio of malicious to normal samples.

The algorithm picks multiple random features and for each one of those, it plots the samples in a graph. To determine the likelihood of a sample being an anomaly, the algorithm makes random cuts in the graph. The higher the number of cuts necessary to isolate a sample, the higher the likelihood of that sample being an anomaly.

The model was run using the hyper-parameters specified in the appendix. These were obtained by doing a 5-fold cross validation over hundreds of various parameter combinations. Note that we did not use an independent dataset for cross-validation and therefore these parameters are optimal for our dataset but may not be generalisable to other network intrusion datasets.

Isolation Forest and other offline models are implemented using Scikit Learn. The experiments with Isolation Forests and Gradient Boosting were run on a dedicated server using an AMD Ryzen 5, 6-core CPU operating at 3.6GHz.

### C. Gradient Boosting

Gradient Boosting is an ensemble of decision trees that iterates over the dataset while optimising a loss function. It has been shown to perform very well for anomaly detection in other domains e.g. the detection of credit card fraud [28].

### D. Kitsune

This model is an ensemble of autoencoders which is trained in an unsupervised manner [12]. The published implementation is in native Python which runs on a single-thread.

This is a notable limitation, as it was only able to process 1 million samples in 4 hours on the AMD Ryzen CPU and the complexity did not scale in a linear fashion.

## VIII. Results

### A. Dataset Processing

The processing rate measured during our experiments was stagnating around *300,000* packets per second on a Ryzen 9 5950X 16 core processor, when processing packets read directly from the packet capture files provided by the dataset authors. An example of the measured throughput in packets per second is plotted in Figure 6.

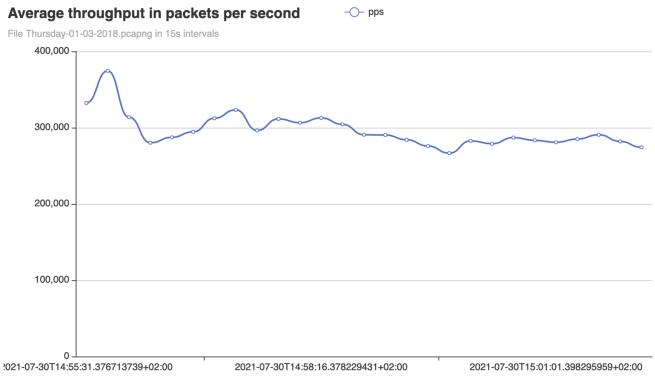


Fig. 6: Packet ingestion performance

To accelerate the processing with Netcap, we only loaded the connection decoder and disabled the TCP stream reassembly to improve the performance. Conversion to connection audit records reduced the dataset file size drastically, for example a *62G* packet capture file with merged captures for a single day, resulted in *1.4G* of uncompressed, labelled connection audit records in CSV format. The entire dataset consists of *436G* of packet captures and is reduced to *7.1G* of data as uncompressed connection audit records from around 40 million bi-directional connections. We used a M.2 Samsung 970 Evo Plus solid state drive to achieve high throughput rates for writing data to disk, as this often poses a bottleneck when dealing with large amounts of data.

### B. Shallow Learning

We evaluated Isolation Forest and Gradient Boosting using the enriched network flow data and the connection audit records since they contain 80 features versus 28 respectively. Higher-dimensional data allows models like Gradient Boosting to learn complex relations between these features and leads to an increase in the detection rate as long as the features are significant.

The used models are from the Scikit Learn Python library. These implementations are not designed to train on large amounts of data and we were therefore not able to run this set of experiments on the entire dataset. Instead we ran the models on individual days. This simplifies the learning task for the models since they are exposed to a single attack type during training and testing, with the exception of one day (Fri-23-02-2018) that includes both the brute force and injection attack types.

All offline model experiments were run on the dataset without any address information to facilitate the comparison to the DNN results. The list of features and the model parameters are available in the Appendix.

When splitting the individual days into train and test sets for the offline models, it can happen that the testing split contains only benign samples to classify, which would lead to a perfect F1 score. In order to avoid this, the train/test split was done after shuffling the dataset. For consistency, we made sure to remove any features that contain time information, such as timestamps.

#### 1) Isolation Forest

In table I we can see that Isolation Forest achieves a high F1 score on a majority of days. It is apparent that on days with a higher attack ratio, the F1 score drops. This is due to Isolation Forest's design, and this drawback is recognised by the authors of the algorithm [6]. In practice this can be alleviated by training on a dataset that contains few anomalies.

Overall, Isolation Forest performed extremely well given the simplicity of the algorithm and was able to achieve an F1 score of over 0.95 on most days. We find that it is a strong contender for detecting Brute-force, DoS, DDoS, and Bot attacks. Additionally Isolation Forest has a linear time complexity and a low constant memory requirement which makes it viable for classifying large amounts of events.

Unlike Gradient Boosting, Isolation Forest is a simple algorithm and we can see that it fails to detect complex attacks such as the ones carried out on infiltration days. Indeed days 28/02 and 01/03 have an F1 score of 0.77 and 0.56 respectively.

Although the model is unsupervised, it is not the best choice be for a Intrusion Detection System (IDS) since it is an offline model. There exists an incrementally updating variant of the algorithm known as Half-Space Trees [29] which may be a reasonable choice for an IDS, and should be subject to future experiments.

Looking at table II, we see that Isolation Forest has a consistently high F1 score when run on connection audit records. The F1 score is higher compared to the one from the experiments on the enriched network flows for all days except 20/02 and 02/03. This is an interesting result since we would expect the model to perform worse given that the connection audit record has fewer significant features than the enriched network flow data. One possible explanation for this is that the attack ratio for connection audit records is lower then for network flows, making the anomalies easier to isolate. Another complimentary reason is that the connection audit record data was encoded using zscoring whereas the enriched network flow data was fed into the algorithm without prior encoding.

The day 20/02 contains a distributed denial of service attack using the Low Orbit Ion Canon tool, and the day 02/03 had Botnet traffic. The enriched network flow data may have

| **Day** | 14/02 | 15/02 | 16/02 | 20/02 | 21/02 | 22/02 | 23/02 | 28/02 | 01/03 | 02/03 |
|---|---|---|---|---|---|---|---|---|---|---|
| **Attack Labels** | Brute-force | DoS | DoS | DDoS | DDoS | Brute-force | Brute-force | Infiltration | Infiltration | Bot |
| **Attack Ratio (%)** | 5.38 | 0.8 | 12.99 | 7.29 | 12.9 | 0.01 | 0.79 | 11.24 | 28.11 | 3.48 |
| **iForest (F1)** | 0.95 | 0.99 | 0.91 | 0.95 | 0.88 | 0.99 | 0.99 | 0.77 | 0.56 | 0.99 |
| **GBoost (F1)** | 1.0 | 0.99 | 0.99 | 0.99 | 0.99 | 0.99 | 0.99 | 0.68 | 0.79 | 0.99 |
| **GBoost * (F1)** | 0.94 | 0.99 | 0.87 | 0.93 | 0.87 | 0.99 | 0.99 | 0.89 | 0.72 | 0.96 |

TABLE I: Offline models run on enriched network flows without address information (* = pruning)

| **Day** | 14/02 | 15/02 | 16/02 | 20/02 | 21/02 | 22/02 | 23/02 | 28/02 | 01/03 | 02/03 |
|---|---|---|---|---|---|---|---|---|---|---|
| **Attack Labels** | Brute-force | DoS | DoS | DDoS | DDoS | Brute-force | Brute-force | Infiltration | Infiltration | Bot |
| **Attack Ratio (%)** | 0.48 | - | 0.59 | 4.61 | 2.74 | 0.000035 | 0.000048 | 1.06 | 2.1 | 1.6 |
| **iForest (F1)** | 0.99 | - | 0.98 | 0.91 | 0.95 | 0.99 | 0.99 | 0.97 | 0.96 | 0.96 |
| **GBoost (F1)** | 0.99 | - | 0.99 | 0.99 | 0.99 | 0.99 | 0.99 | 0.99 | 0.98 | 0.99 |
| **GBoost * (F1)** | 0.99 | - | 0.99 | 0.95 | 0.97 | 0.99 | 0.99 | 0.99 | 0.98 | 0.98 |

TABLE II: Offline models run on connection audit records without address information (* = pruning)

| **Day** | 14/02 | 15/02 | 16/02 | 20/02 | 21/02 | 22/02 | 23/02 | 28/02 | 01/03 | 02/03 |
|---|---|---|---|---|---|---|---|---|---|---|
| **Attack Labels** | Brute-force | DoS | DoS | DDoS | DDoS | Brute-force | Brute-force | Infiltration | Infiltration | Bot |
| **Attack Ratio (%)** | 0.0048 | 0.54 | 0.0059 | 0.046 | 0.027 | 0.000037 | 0.000049 | 0.011 | 0.21 | 0.016 |
| **F1** | 0.65 | 0.51 | 0.59 | 0.65 | 0 | 0.68 | 0.68 | 0.53 | 0.45 | 0.43 |

TABLE III: Kitsune run on the first 1 million lines of connection audit records using the address information

| **Day** | **Labels** | **Network Flow** | | **Connection Audit Records** | | **Flags** | |
|---|---|---|---|---|---|---|---|
| | | samples | % malicious | samples | % malicious | FIN | RST |
| 14/02 | Brute-force | 7,083,083 | 5.38 | 1,351,639 | 0.48 | 41,836 | 1,273,384 |
| 15/02 | DoS | 6,617,761 | 0.8 | 458,568 | 0 | 41,480 | 1,144,304 |
| 16/02 | DoS | 8,520,652 | 12.99 | 2,389,314 | 0.59 | 41,017 | 1,457,712 |
| 20/02 | DDoS | 7,948,749 | 7.29 | 3,193,359 | 4.61 | 45,951 | 1,685,917 |
| 21/02 | DDoS | 9,603,223 | 12.99 | 5,380,173 | 2.74 | 46,840 | 2,607,408 |
| 22/02 | Brute-force | 8,179,616 | 0.01 | 5,481,910 | 0.000035 | 51,418 | 1,850,717 |
| 23/02 | Brute-force | 7,928,197 | 0.79 | 4,002,952 | 0.000048 | 52,119 | 1,741,312 |
| 28/02 | Infiltration | 613,071 | 11.24 | 5,953,596 | 1.06 | 3,805 | 139,251 |
| 01/03 | Infiltration | 331,101 | 28.11 | 5,961,260 | 2.1 | 1,951 | 65,904 |
| 02/03 | Bot | 8,217,203 | 11.24 | 5,732,010 | 1.6 | 38,574 | 2,308,407 |

TABLE IV: Attack Ratio for different dataset types

performed better on these days simply due to the additional features, such as the DstPort and Protocol pair frequency.

*2) Gradient Boosting*

We used Gradient Boosted Decision Trees for binary classification. As seen in Table I it performed very well, achieving an F1 score close to 1 on all days except the infiltration day 28/02. This is likely because the model is using the optimal hyperparameters which were discovered during 5-fold cross validation.

This indicates that the model is overfitting the dataset and as a consequence might generalise poorly i.e. it may not perform as well with real-world data. Luckily, we can tweak the number of trees that are used in the ensemble, known as the number of estimators. Another interesting parameter to consider is the contribution that each new decision tree adds to the overall classification decision, known as the learning rate. Lowering the learning rate theoretically decreases the F1 score but increases generalisability. In addition to this, we can also prune the decision trees to discard any leaf nodes that do not meet a given cost complexity threshold. When enabled, pruning mitigates overfitting, as seen in Table I.

To further reduce training time and mitigate overfitting, this model made use of early stopping. It should be noted that Gradient Boosting is known to be fairly robust to overfitting [30] since the decision trees that make up the ensemble are short, with a depth of 2 or 3 in our experiments.

### C. Deep Learning

The following section will present the results of our experiments that made of deep learning algorithms.

*1) Kitsune*

The Python implementation of this ensemble of auto-encoders would process around *250,000* samples per hour.

Given that most connection audit record collections for single days in the dataset, contain around 5 million samples we can estimate a 20 hour runtime per day. Our preliminary run on the first 1 million lines of each file, in table III, shows that the model is performing poorly and we suspect this occurs likely due to under-exposure to anomalies. Future experiments should run the Kitsune experiments on the entire dataset, for a fair comparison.

*2) Deep Neural Network*

The deep neural network created using Tensorflow performed well in terms of training time due to running on the GPU. This can be seen in the time column of the result tables. The short processing times allowed for series of experiments to be executed overnight and the results being ready for analysis the next day. The results of training a model for each day of the dataset are listed in Table V. The day *15/02* did not contain any attacks in the provided packet captures, and could therefore not be evaluated. The following observations are made: for the days *22/02* and *23/02*, detection of malicious activity seemed not possible, due to the very low number of malicious examples. Even the class weights were not able to fix this heavy imbalance. Likely additional features would be required on the connection audit records to make a successful distinction in this case, or other techniques such as oversampling should be considered to increase the amount of malicious events. In general, there does not seem to be a configuration in which all attacks are equally well detected. For the brute force attacks on the *14/02*, the best performing configurations were using address information, but the absence of address information did not seem to affect the models prediction performance much. For the denial of service attack day *16/02*, we observe that both with and without address information and tweaked class weights, an F1 score of *0.99* is reached, the run with address information and dropout layer, and the one with tweaked class weight even scored 1. The pattern created by a denial of service attack seems to be detectable with a very high certainty, even with the absence of address information. The days with the distributed variant of this attack, *20/02* and *21/02*, do also deliver very high detection rates. The two infiltration attack days *28/02* and *01/03* have a quite similar detection rate across the different variants. The scores seem generally high when using address information, but do perform slightly worse in case no address information is used. The botnet attack day on *02/03* delivered a minimum F1 score of *0.93* with address information, and a consistent F1 score of *0.79* without address information. We observe that this model delivered several false positives, but does seem to be able to detect the general pattern of botnet activity in the majority of cases. The combination of *ReLu* activation and *MinMax* normalisation delivered poor results across all experiments, while the use of *LeakyReLu* activation and *Zscore* normalisation delivered the best results.

The experiments on the entire dataset are displayed in Table VI. The highest F1 score of *0.95* is achieved when using the address information, with absence of address information the score drops to *0.87*, which indicates the presence of several false classifications, while still detecting the majority of malicious connections. The class weights using the standard procedure seem to lead to a drastic over prioritisation of the attack class, and as a consequence yield bad scores due to a high amount of false classifications. Our applied tweaking seems to help there, both when using address information and without it. Overall we observe that the absence of address information affects the achieved score, but does not completely prevent the algorithm from making useful predictions. This is relevant in case knowledge transfer to a different network topology is desired, for example when a vendor of security software wants to train a model on a dataset and ship the model to customers for deployment in network environments that differ from the one used in the dataset. However, in order to achieve the highest accuracy in detection rate and the least false alarms, the use of address information should be considered, even when this requires training the model with a dataset recorded in the target network.

Results for the experiments with multi-class classification are shown in Table VII. In the confusion matrix in figure 7 we can see that even without using address information predictions include 4 of 6 classes in the dataset (normal, denial-of-service, infiltration and botnet), which is a similar result compared to the variant with address information. The training phase metrics for the run with address information are displayed in figure 8. Two classes, bruteforce and injection, are heavily underrepresented in the testing split, as they appear only between 4-1751 times in the dataset with around 40 million connections, which is likely the reason these are never detected successfully. Overall the detection performance still seems very high when using multiple classes.

The batch size is the number of samples that will be propagated through the neural network. The estimate of the gradient will be less accurate, the smaller the value is that was chosen for the batch size. In the experiments for determining the optimal batch size, we observed the configuration with 1024 samples per batch to be performing the best. Our baseline experiments have been executed using a batch size of 2048. We also observed that the batch size has a big impact on computational performance, with larger batch sizes processing much faster, ranging from around 30 seconds per epoch on the entire dataset with a value of 2048, and up to 30 minutes per epoch when letting Tensorflow set this value for us and setting it to a very small number of samples (4-16).

Overall, the DNN predictions gave useful results, in case sufficient instances of the attack class were available during training. However, there was no model that was entirely free of false positives or false negatives, and thus we conclude that this method still requires human supervision and manual alert analysis, and is not suited for automated action at this moment. Further feature additions to the connection audit records could help to solve this problem, as well as further alert enrichment and post processing, before presenting the alert to an analyst.

| Day | 14/02 | 15/02 | 16/02 | 20/02 | 21/02 | 22/02 | 23/02 | 28/02 | 01/03 | 02/03 |
|---|---|---|---|---|---|---|---|---|---|---|
| **Attack Labels** | Brute-force | DoS | DoS | DDoS | DDoS | Brute-force | Brute-force | Infiltration | Infiltration | Bot |
| **Attack Ratio (%)** | 0.48 | - | 0.59 | 4.61 | 2.74 | 0.000035 | 0.000048 | 1.06 | 2.1 | 1.6 |
| **A (F1)** | 0.99 | - | 0.99 | 0 | 0.94 | 0 | 0 | 0.97 | 0.96 | 0.93 |
| **A + CW (F1)** | 0 | - | 0.99 | 0.94 | 0.94 | 0 | 0 | 0.87 | 0.88 | 0.90 |
| **A + TCW (F1)** | 0.99 | - | 1 | 0.94 | 0.94 | 0 | 0 | 0.97 | 0.97 | 0.94 |
| **A + DL (F1)** | 0.98 | - | 1 | 0 | 0.94 | 0 | 0 | 0.96 | 0.95 | 0.93 |
| **A + RM (F1)** | 0 | - | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.79 |
| **NA (F1)** | 0.97 | - | 0.99 | 0.94 | 0.94 | 0 | 0 | 0 | 0.81 | 0.79 |
| **NA + CW (F1)** | 0 | - | 0.99 | 0.94 | 0.94 | 0 | 0 | 0.64 | 0.60 | 0.79 |
| **NA + TCW (F1)** | 0.98 | - | 0.99 | 0.94 | 0.94 | 0 | 0 | 0.83 | 0.80 | 0.79 |
| **NA + DL (F1)** | 0.97 | - | 0.99 | 0.94 | 0.94 | 0 | 0 | 0.83 | 0 | 0.79 |
| **NA + RM (F1)** | 0 | - | 0 | 0 | 0.93 | 0 | 0 | 0 | 0 | 0 |

TABLE V: DNN on single days, binary classification for classes normal and attack, A=Addresses used, NA=No Addresses used, CW=Class Weights, TCW=Tweaked Class Weights, '-' indicates that there were no attacks in the provided packet captures

| Variation | F1 | Time (s) | Epochs | False Positives | False Negatives |
|---|---|---|---|---|---|
| **A** | 0 | 685 | 4 | 0 | 170143 |
| **A + CW** | 0.86 | 803 | 5 | 49392 | 2215 |
| **A + TCW** | 0.95 | 941 | 6 | 11480 | 5531 |
| **A + DL** | 0 | 692 | 4 | 0 | 169952 |
| **A + RM** | 0 | 799 | 4 | 0 | 169771 |
| **NA** | 0.86 | 782 | 5 | 23668 | 21367 |
| **NA + CW** | 0.55 | 648 | 4 | 254431 | 5008 |
| **NA + TCW** | 0.87 | 1025 | 7 | 27018 | 18269 |
| **NA + DL** | 0.87 | 1018 | 7 | 23851 | 20086 |
| **NA + RM** | 0 | 736 | 4 | 0 | 169750 |

TABLE VI: DNN on entire dataset, binary classification for classes normal and attack, dnn size: 16 - 32 - 16, batch size 1024, A=Addresses used, NA=No Addresses used, CW=Class Weights, TCW=Tweaked Class Weights, DL=Dropout layer used, RM=ReLu activation and MinMax normalisation used

| Variation | F1 | Time (s) | Epochs | False Positives | False Negatives |
|---|---|---|---|---|---|
| **A** | 0.95 | 741 | 5 | 8097 | 6 |
| **A + CW** | 0.56 | 996 | 7 | 95189 | 8513 |
| **A + DL** | 0.94 | 748 | 5 | 8318 | 6 |
| **A + RM** | 0 | 776 | 4 | 0 | 77456 |
| **NA** | 0.94 | 1172 | 9 | 8378 | 11 |
| **NA + CW** | 0.91 | 952 | 7 | 13644 | 4 |
| **NA + DL** | 0.94 | 721 | 5 | 8443 | 7 |
| **NA + RM** | 0 | 690 | 4 | 0 | 77351 |

TABLE VII: DNN multi-class classification experiments on entire dataset, dnn size: 16 - 32 - 16, batch size 1024, classes: normal, denial-of-service, bruteforce, injection, infiltration and botnet, A=Addresses used, NA=No Addresses used, CW=Class Weights, DL=Dropout layer used, RM=ReLu activation and MinMax normalisation used

## IX. Discussion

### A. Limitations

When evaluating the performance of an algorithm on a dataset, the structure of the dataset has to be taken into account. In this case, individual attack classes are isolated on a single day, which helps to observe the efficiency of an algorithm for detecting a specific attack class. This might lead to a model performing badly when multiple different attacks appear simultaneously, because it expects them to occur only in isolation, as seen previously in the training data.

The audit record type that was chosen is in theory time-independent, and a single connection can be moved around the time axis. However, there are situations where multiple connections are associated to a malicious action, and their order could be used as an additional indicator. The current implementation does not make use of this, as the data points are shuffled before being fed to the classifier, which helps to

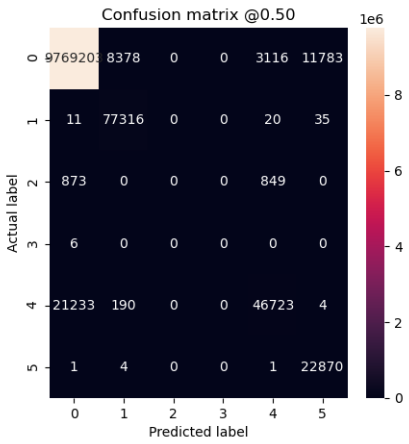


Fig. 7: Confusion matrix for multi-class classification on entire dataset without address information, labels: 0: normal, 1: denial-of-service, 2: bruteforce, 3: injection, 4: infiltration, 5: botnet

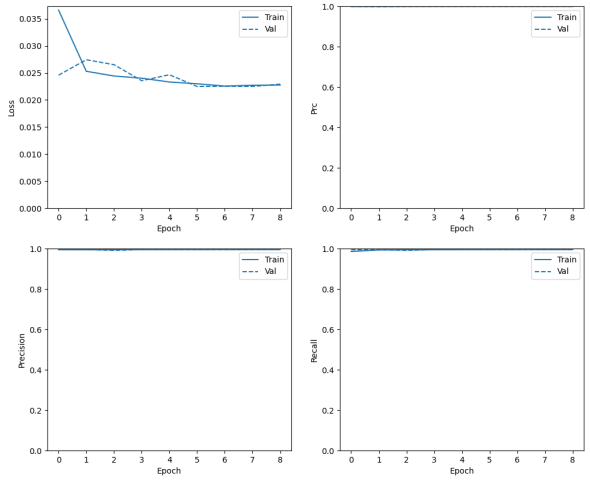


Fig. 8: Training metrics for multi-class classification on entire dataset without address information

create an equal distribution of normal and benign connections in the training and testing split. As long as this is applied both in the experiment setup and in production, we expect similar results.

In addition, the use of time information could lead to unexpected behaviour, for example when a model associates Wednesdays with denial of service attacks, because in the training dataset those attacks only occurred on this day. The same could happen for a specific time of the day, or other attributes. Those hidden correlations could cause unintended bias for the model and could degrade prediction performance. The DNN in the sequential dense layer configuration discards all time related information, but for future work with Long Short Term Memory (LSTM) layers such side effects must be considered. When using other audit record types, for example those for TCP packets, more care has to be given here because the order of packets matters for the correct interpretation, due to the stateful nature of the underlying protocol. A further problem is the delay due to batch processing: the batch size (e.g: 1024 connections) creates a delay in the generated alerts, which could be exploited by an attacker during periods when there is not much network traffic.

The impact of concept drift, which is the change of statistical properties of the testing data [31] must be further assessed for the connection audit records as well. As a consequence of concept drift, the performance of the classifier is expected to decrease over time. Retraining the model will be required in a regular interval in case the environment is non stationary [32]. This greatly depends on the target environment: while industrial systems are generally more stationary and change less often, corporate network where employees can bring their own devices change patterns frequently. We applied shuffling to prevent spatial bias and enforce a similar distribution of the positive and negative classes, and chose the connection audit record abstraction, because it can be moved on the time axis in order to prevent time bias. In addition, the deep neural network discards all time information. When choosing a classifier for a target environment, it should therefore be evaluated if the stability of the classifier over time is more important than its detection performance.

### B. Visualisation

Data gathered from network connections is very abstract machine data, and in order to make it understandable to a human analyst, different types of visualisation are commonly used nowadays.

To make the dataset exploration more visual and interactive, we used the industry standard *elasticsearch* and *kibana* stack to visualise the collected data as charts and data plots. Future versions could also use this platform for tracking of generated alerts. In addition, graphical link based analysis with the *Maltego* software suite was performed, to gather insight into the characteristics of the attacks, as well as to perform exploratory data analysis.

### C. Alerting

As the number of connections related to an attack can vary from a dozen to several thousand ones, we identified alert *debouncing* and *denoising* as two important aspects. With *debouncing*, we refer to the reduction of the number of alerts produced, for example by grouping alerts based on the affected hosts, or fetching alert summaries in a regular interval, instead of using a real-time feed. With *denoising*, we refer to the reduction of noise caused by alerts considered non-important. Filtering or scoring mechanisms could be applied in order to decrease the number of alerts presented to an analyst, or a voting based ensemble model could be used to discard events without a sufficient number of votes. In general, great care

has to be taken for the delivery of alerts to an analyst. Too many false positives will cause alert fatique, meaning that the analyst will start to ignore alarms because he suspects them to be incorrect. This is of course dangerous and unwanted, and as such advanced network intrusion detection solutions should further process generated alerts to ensure they are indeed worth notifying an analyst.

## X. Conclusion

Even in absence of computationally expensive generated features from the connection audit records, as well in absence of address information features, solid detection rates have been achieved in the experiments. For this, basic features about the volume of transferred data, destination port information as well as protocol names of each involved layer have been used, to construct summary structures that can express behavioural patterns on the network, while having a relatively small footprint on disk and in memory. Adding the TCP specific flag counters to the connection audit records further improved the detection performance. Regarding packet processing and audit record generation on commodity hardware, throughput rates of around *300,000* packets have been measured in our experiments. The generated data as connection audit records requires less storage space (7.1G) compared to the provided network flow data (31G) and compresses extremely well, with over *70%* possible size reduction. The evaluated algorithms that delivered the best classification performance in our experiments are Gradient Boosting and the Deep Neural Network with sequential dense layers. The best performance in regard to training speed was the DNN due to execution on a GPU. Evaluation of the Kitsune framework has been limited due to time constraints, and should therefore be repeated on the entire available data.

Use of a Graphical Processing Unit (GPU) for accelerating machine learning operations and parallelisation to increase performance of the feature extraction, are essential for processing large amounts of data. Common tooling in the data science and packet processing area is unfortunately often single threaded and as a consequence unsuited for processing such large scale datasets.

Overfitting of certain models can be mitigated to make them generalisable, for example the removal of address information demonstrated in our experiments, and the use of discussed techniques like early stopping. The DNN experiments indicated that knowledge transfer to another network topology seems possible, as the detection results are still high even when removing address information. The Gradient Boosting experiments have shown that models are still prone to overfitting without address information and when using early stopping and therefore must be evaluated on real-world data in future research. Efficiency in detecting attacks is highly affected by the imbalanced nature of the data. Some days might contain no malicious actions at all, while others may feature attack types like denial of service that in some cases drastically change the ratio of the negative and positive classes. The used algorithms expect a similar distribution of the positive to the negative class in the training and testing dataset. This is a potential problem, since this ratio can be influenced by an attacker. Ideally, features should be used that are hard to manipulate by an attacker. The bias impaired on the model due to this data distribution assumption, and the implications for the use of such algorithms for network security monitoring, need to be further explored. Shallow and deep learning algorithms also suffer from performance decay over time, due to concept drifting or the introduction of new attack variations that require retraining the model or the addition of new features. This has to be considered for the deployment of such systems.

A challenge for the automated analysis of network traffic besides performance limitations is the fact that not every anomaly is malicious, as in many cases anomalies are caused by equipment malfunction or misconfigurations, rather than by an attacker. Inversely not every malicious connection is anomalous. Attackers can try to attack the distributional assumptions made by machine learning models, much in the same way attackers obfuscate known attacks by modifying their signatures to bypass signature-based systems. Furthermore, attackers can model normal network behaviour that was observed with passive network sniffing, to avoid detection and blend in with what is known to be normal traffic.

Address information might not be required or not desired for the algorithm that makes the decision, but is still very important information for the analyst. The online algorithm we tested performed generally poorly, both in regard to detection efficiency and performance of the provided implementation. Lastly, we would like to reason about the need to detect every attack for uncovering network intrusions, and the impact of false alarms on the perception and behaviour of an analyst. False alarms will cause the analyst to ignore events if they occur frequently, which is undesirable for the operation of security monitoring, as it decreases overall system efficiency. In order to detect an ongoing incident, a single alarm that identified a component of the attack chain might be enough, to have forensic experts follow up on the alert and discover the breach. Overall, the detection performance observed in the experiments was very high, and would certainly provide value to a threat analyst, even if human intervention for the alert analysis is required. The key goal for a network intrusion detection system is to reduce the workload imposed on human analysts due to manual artifact analysis, especially when dealing with huge amounts of data. We see great potential for the application of the analysed algorithms, and plan to asses their performance in future deployments on real-world data. Machine learning and deep learning is not subject to human bias, but extremely sensitive to the data it was trained on. As such, data pollution by an attacker remains a serious threat for the effectiveness of such systems. The limitations imposed by the lack of explainability for some algorithms, can be reduced by enriching alerts with additional information about the affected systems, to help an analyst to make a quick but well informed decision. Regarding the choice of algorithm for the detection task, we conclude that each algorithm delivers a different detection performance for different attack types,

and as a consequence must be chosen with care. If possible, a simpler algorithm that requires less computational resources should be preferred for the detection of attacks with significant statistical properties, such as denial of service or brute force attacks, instead of using a complex algorithm like a deep neural network.

## XI. Future Work

Future research includes completing the alert pipeline and exploring alert analysis in *Maltego* / *Elastic*. In addition, further research and more experiments with unsupervised algorithms should be conducted, as those would be better suited for the deployment requirements of nowadays network infrastructure.

Feature collection could also make use of the Extended Berkeley Packet Filter (eBPF) provided by the Linux kernel, in order to ease the computational pressure on the intrusion detection system imposed due to packet processing and improve performance. Indeed it has been shown that a flow based network intrusion detection system can be implemented entirely in eBPF resulting in a 20% performance increase over user-space programs [33].

After identifying algorithms that show promising results for a particular attack type, combining them in an ensemble fashion to compensate the weaknesses of individual models should be considered, as an ensemble is superior to individual approaches [11]. Ensemble methods can also help to scale for large datasets [2].

Feature engineering could make use of address information or subnets to define certain trust or network groups, so that address information can be used for classification, but is not bound to specific addresses and can be configured prior to deployment. Such groups could include internal or external hosts, administrator, user or guest user Internet Protocol (IP) addresses, or internal services, based on their relevance.

Further feature engineering could increase the likelihood of detecting anomalous behaviour for a sample, for example by using generic indicators for the creation of new features, such as: if a connection is made from a workstation outside office hours, if the connection is towards a blacklisted country, or if large volumes of data are leaving the internal network, for example during data exfiltration. Extracted features could also include hashes that identify the initiators and providers of encrypted communications, such as the TLS client and server fingerprints (Ja3) and SSH client and server fingerprints (HASSH) mechanisms developed by salesforce. Also the use of non standard port numbers for web services or the use of Secure Hypertext Transfer Protocol (HTTP) requests towards an IP address instead of a host name could be used as a trigger to raise an alert. Deep packet inspection could be used for further feature enrichment and also assist in the creation of generic detection rules, for example consider checking whether the actual file content type matches the advertised one, or checking if an Uniform Resource Locator (URL) parameter contains shell commands using a regular expression.

Using different layer types for the deep neural network, that can make sense of time should also be considered. As concurrent processing can potentially result in output being generated in a randomised order, further experiments should assess the impact of this on the quality of the detection accuracy. Furthermore, optimizers other than the used *adam* algorithm should be evaluated and their performance compared. The monitored value to determine the point for early stopping could also be changed, for example to using the *precision-recall curve* instead of the *loss* value used in our experiments. Furthermore, future experiments should asses an acceptable loss value for the scenario and determine if our chosen value is optimal or could be improved.

Alerts should be emitted in an industry standard format, such as the Malware Information Sharing Platform (MISP) or Structured Threat Information eXpression (STIX) standards, in order to enable sharing indicators of compromise with external systems [22].

The Kitsune ensemble of autoencoders and the decision trees could also be implemented via Tensorflow to increase sample throughput. Also different audit record types should be evaluated in isolation or in combination with connection audit records, including but not limited to DNS, HTTP, ICMP and Secure Shell Protocol (SSH) records.

Apart from evaluation on datasets, a case study with real-world traffic could also deliver valuable insights, especially regarding deployment challenges and performance of the solution over time under the influence of concept drift. The DNN experiments could be extended with using LSTM layers, to observe the impact of time as an additional factor. This would allow for the use of packet based audit records, such as those for TCP, as the order here is crucial for the correct interpretation. To combat the heavy class imbalance for some attack types, also oversampling could be considered. For encoding categoricals, learned embedding could be applied in order to compare the results to the enumeration based approach. Furthermore it would be interesting to see how the detection performance changes when using one-hot encoding. The Keras functions *evaluate*, *predict* and *fit* for training and evaluating the models have parameters for parallel processing, namely *workers* and *use_multiprocessing*. Possible performance improvements with these configuration options should be explored in future experiments. The code used to run the DNN experiments should be extended to use K-fold cross validation, to ensure the results are not impacted by an uneven distribution in the class ratio of the positive and negative classes for the train and test dataset splits, and the final result score is averaged over multiple runs. We note however that due to applying shuffling and measuring the distribution of classes, we ensured that the ratio is equal in our experiments, and have observed consecutive runs of the same experiment to produce similar results. To allow for incremental updates of the DNN, the Keras batch training API should be used for future versions.

## References


[1] Mahendiran, A., Appusamy, R., and S, K. "Intrusion Detection and Prevention System: Tchnologies and Challenges". In: *International Journal of Applied Engineering Research* 10 (Jan. 2015), pp. 1–12.

[2] Bhattacharyya, D. K. and Kalita, J. K. *Network Anomaly Detection: A Machine Learning Perspective*. Chapman amp; Hall/CRC, 2013. ISBN: 1466582081.

[3] Veeramachaneni, K. et al. "AIˆ 2: training a big data machine to defend". In: *2016 IEEE 2nd International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS)*. IEEE. 2016, pp. 49–54.

[4] Lazarevic, A. et al. "A Comparative Study of Anomaly Detection Schemes in Network Intrusion Detection". In: vol. 3. May 2003. DOI: 10.1137/1.9781611972733.3.

[5] Emmott, A. F. et al. "Systematic construction of anomaly detection benchmarks from real data". In: *Proceedings of the ACM SIGKDD workshop on outlier detection and description*. 2013, pp. 16–21.

[6] Liu, F. T., Ting, K. M., and Zhou, Z.-H. "Isolation forest". In: *2008 eighth ieee international conference on data mining*. IEEE. 2008, pp. 413–422.

[7] Naseer, S. et al. "Enhanced network anomaly detection based on deep neural networks". In: *IEEE access* 6 (2018), pp. 48231–48246.

[8] Hendrycks, D., Mazeika, M., and Dietterich, T. "Deep anomaly detection with outlier exposure". In: *arXiv preprint arXiv:1812.04606* (2018).

[9] Amarasinghe, K., Kenney, K., and Manic, M. "Toward explainable deep neural network based anomaly detection". In: *2018 11th International Conference on Human System Interaction (HSI)*. IEEE. 2018, pp. 311–317.

[10] Nguyen, A., Yosinski, J., and Clune, J. "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 427–436.

[11] Mukkamala, S., Sung, A. H., and Abraham, A. "Intrusion detection using an ensemble of intelligent paradigms". In: *Journal of network and computer applications* 28.2 (2005), pp. 167–182.

[12] Mirsky, Y. et al. "Kitsune: An ensemble of autoencoders for online network intrusion detection. arXiv 2018". In: *arXiv preprint arXiv:1802.09089* ().

[13] Sharafaldin, I., Lashkari, A. H., and Ghorbani, A. A. "Toward generating a new intrusion detection dataset and intrusion traffic characterization." In: *ICISSp*. 2018, pp. 108–116.

[14] Richard, B. *The Practice of Network Security Monitoring - Understanding Incident Detection and Response*. No Starch Press, July 2013. ISBN: 978-1-59327-509-9.

[15] Michael W., L. *Network Flow Analysis*. No Starch Press, June 2010. ISBN: 978-1-59327-203-6.

[16] Stephen, N. and Judy, N. *Network Intrusion Detection, Third Edition*. New Riders Publishing, August 28, 2002. ISBN: 0-73571-265-4.

[17] Gu, J. "An Effective Intrusion Detection Model Based on Pls-Logistic Regression with Feature Augmentation". In: Jan. 2020, pp. 133–140. ISBN: 978-981-33-4921-6. DOI: 10.1007/978-981-33-4922-3_10.

[18] Chris, M. *Network Security Assessment, 3rd Edition*. O'Reilly Media, Inc., December 2016. ISBN: 9781491910955.

[19] James, F. *Attacking Network Protocols - A Hacker's Guide to Capture, Analysis, and Exploitation*. No Starch Press, December 2017. ISBN: 9781593277505.

[20] Mieden, P. *NETCAP - A framework for secure and scalable network traffic analysis*. 2021. URL: https://github.com/dreadl0ck/netcap.

[21] Zheng, A. and Casari, A. *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists*. 1st. O'Reilly Media, Inc., 2018. ISBN: 1491953241.

[22] Collins, M. *Network Security Through Data Analysis, 2nd Edition*. O'Reilly Media, Inc., September 2017. ISBN: 9781491962848.

[23] Sateesh, S. *Have you asked why F1-Score is a Harmonic Mean(HM) of Precision and Recall*. 2018. URL: https://medium.com/@srinivas.sateesh/have-you-asked-why-f1-score-is-a-harmonic-mean-hm-of-precision-and-recall-febc233ce247.

[24] Brownlee, J. *Loss and Loss Functions for Training Deep Learning Neural Networks*. 2019. URL: https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/.

[25] *Classification on imbalanced data*. 2021. URL: https://www.tensorflow.org/tutorials/structured_data/imbalanced_data.

[26] Mieden, P. and Partarrieu, P. *Masterthesis experiment code and logs*. 2021. URL: https://github.com/dreadl0ck/masterthesis.

[27] Tom, H., Yehezkel S., R., and Itay, L. *Learning TensorFlow*. O'Reilly Media, Inc., August 2017. ISBN: 9781491978511.

[28] Psychoula, I. et al. "Explainable Machine Learning for Fraud Detection". In: *arXiv preprint arXiv:2105.06314* (2021).

[29] Tan, S. C., Ting, K. M., and Liu, T. F. "Fast anomaly detection for streaming data". In: *Twenty-Second International Joint Conference on Artificial Intelligence*. 2011.

[30] URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html.

[31] Cavallaro, L. "When the Magic Wears Off: Flaws in ML for Security Evaluations (and What to Do about It)". In: Burlingame, CA: USENIX Association, Jan. 2019.

[32] Jordaney, R. et al. "Transcend: Detecting Concept Drift in Malware Classification Models". In: *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 625–642. ISBN: 978-1-931971-40-9. URL: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/jordaney.

[33] Bachl, M., Fabini, J., and Zseby, T. "A flow-based IDS using Machine Learning in eBPF". In: *arXiv preprint arXiv:2102.09980* (2021).

## APPENDIX

### A. Model Parameters

To allow for reproduction of our results, the following section lists the parameters for the various algorithms we have tested.

**Isolation Forest** was run with 80 estimators, meaning that the forest was made of 80 trees. The contamination rate was set for every individual file since this model is offline and the dataset is known a priori, but the model performed equally as well when the proportion of outliers was determined as in the original paper.

**Gradient Boosting** was configured differently for network flow data and audit records since decision tree can overfit on dataset with lots of features but few samples. For enriched network flow data we used a learning rate of *0.05*, in combination with using the Friedman Mean Squared Error function to measure the quality of a split. The maximum depth of the trees was 3. We used $\sqrt{n\_features}$ which results in 9 features for the network flow data. For audit records, gradient boosting was configured to use a learning rate of *0.01*, a max tree depth of 2, and a single estimator. For experiments where pruning is enabled, we set the alpha cost complexity to 0.005 such that the subtree with the largest cost complexity smaller then alpha was selected. For all data types, early stopping was configured such that the training would cease if the loss decreased by less then $10^-3$. We set aside 0.1% of the training data as a validation set for early stopping.

**Kitsune's** feature mapper was trained on the first 10k samples and the anomaly detector was trained on the next 10k samples. The resulting 80k samples are be used for evaluation. Notice how this model uses 80% of the dataset for evaluation rather than the usual 20 or 25%.

**The Deep Neural Network** used was created with Keras and Tensorflow, and executed using *libcudnn* on a GEFORCE RTX 3090 GPU. The baseline model consists of two fully connected (dense) wrap layers with 16 neurons, 16 core layers with 32 neurons, followed by two additional wrap layers with 16 neurons and a final output layer with one neuron per class. We utilised sequential, fully connected (dense) layers. For the experiments with binary classification, we chose the *binary_crossentropy* loss function in combination with a *sigmoid* activation function on the output layer. Multi-class classification made use of the lossfunction *categorical_crossentropy* and the *softmax* activation. We made use of the *adam* optimizer, and the *LeakyReLu* activation function with an alpha of *0.3* for all layers except the output layer. When using the *relu* activation function, it is used in combination with *minmax* normalisation. The experiments with dropout layers used a dropout rate of *0.3*. The *kernel_initializer* used was always set to 'normal' for all layers. The batch size for feeding data to the model was set to *2048*.

### B. Enriched Network Flow Features

The following features are provided by the enriched network flow data from the CIC dataset authors. We have added the *DstPort-Protocol Pair Frequency* feature.

```
Dst Port
Protocol
DstPort-Protocol Pair Frequency
Flow Duration
Tot Fwd Pkts
Tot Bwd Pkts
TotLen Fwd Pkts
TotLen Bwd Pkts
Fwd Pkt Len Max
Fwd Pkt Len Min
Fwd Pkt Len Mean
Fwd Pkt Len Std
Bwd Pkt Len Max
Bwd Pkt Len Min
Bwd Pkt Len Mean
Bwd Pkt Len Std
Flow IAT Mean
Fwd PSH Flags
Flow IAT Std
Flow IAT Max
Flow IAT Min
Bwd IAT Mean
Fwd IAT Mean
Fwd IAT Tot
Flow Byts/s
Flow Pkts/s
Fwd IAT Std
Fwd IAT Max
Fwd IAT Min
Bwd IAT Tot
Bwd IAT Std
Bwd IAT Max
Bwd IAT Min
Fwd Header Len
Bwd Header Len
Fwd Pkts/s
Bwd Pkts/s
Pkt Len Min
Pkt Len Max
Pkt Len Std
Pkt Len Var
Pkt Len Mean
RST Flag Cnt
FIN Flag Cnt
SYN Flag Cnt
PSH Flag Cnt
ACK Flag Cnt
URG Flag Cnt
Bwd PSH Flags
Fwd URG Flags
Bwd URG Flags
CWE Flag Count
ECE Flag Cnt
Down/Up Ratio
Pkt Size Avg
Fwd Seg Size Avg
Bwd Seg Size Avg
Fwd Byts/b Avg
Fwd Pkts/b Avg
Fwd Blk Rate Avg
Bwd Byts/b Avg
Bwd Pkts/b Avg
Bwd Blk Rate Avg
Subflow Fwd Pkts
Subflow Fwd Byts
Subflow Bwd Pkts
Subflow Bwd Byts
Init Fwd Win Byts
```

```
Init Bwd Win Byts
Fwd Act Data Pkts
Fwd Seg Size Min
Active Mean
Active Std
Active Max
Active Min
Idle Mean
Idle Std
Idle Max
Idle Min
```

### C. Connection Audit Record Features

The following features are provided by the Netcap connection audit records:

```
TimestampFirst
LinkProto
NetworkProto
TransportProto
ApplicationProto
SrcMAC
DstMAC
SrcIP
SrcPort
DstIP
DstPort
TotalSize
AppPayloadSize
NumPackets
Duration
TimestampLast
BytesClientToServer
BytesServerToClient
NumFINFlags
NumRSTFlags
NumACKFlags
NumSYNFlags
NumURGFlags
NumECEFlags
NumPSHFlags
NumCWRFlags
NumNSFlags
MeanWindowSize
```

List of Acronyms

| | |
|---|---|
| **ML** | Machine Learning |
| **DoS** | Denial of Service |
| **DDoS** | Distributed Denial of Service |
| **DNN** | Deep Neural Network |
| **LOF** | Local Outlier Factor |
| **SVM** | Support Vector Machine |
| **GMM** | Gaussian Mixture Model |
| **EGMM** | Ensemble Gaussian Mixture Model |
| **IDS** | Intrusion Detection System |
| **NN** | Neural Network |
| **LSTM** | Long Short Term Memory |
| **TCP** | Transport Control Protocol |
| **TLS** | Transport Layer Security |
| **ICMP** | Internet Control Message Protocol |
| **DNS** | Domain Name Service |
| **Ja3** | TLS client and server fingerprints |
| **HASSH** | SSH client and server fingerprints |
| **SSH** | Secure Shell Protocol |
| **HTTP** | Secure Hypertext Transfer Protocol |
| **HTTPS** | Secure Hypertext Transfer Protocol |
| **MISP** | Malware Information Sharing Platform |
| **STIX** | Structured Threat Information eXpression |
| **AWS** | Amazon Web Services |
| **IP** | Internet Protocol |
| **URL** | Uniform Resource Locator |
| **eBPF** | Extended Berkeley Packet Filter |
| **ROCAUC** | Compute Area Under the Receiver Operating Characteristic Curve |
| **CSV** | Comma Separated Values |
| **GPU** | Graphical Processing Unit |