University of Science and
Technology in Bydgoszcz

**Institute of Telecommunications
and Computer Science**

PhD Thesis

Marek Pawlicki

The Application of Machine Learning in Network Intrusion Detection

Supervisors
Michał Choraś
Rafał Kozik

Bydgoszcz 2020

Contents

1. Introduction

In this section the motivation for this work is given, and the thesis is formulated. The authors publications in the domain pertaining to the dissertation are also disclosed.

1.1. Motivation

The proliferation of communication and commerce over networking architectures has put reliable cyber security solutions in high demand. Cyber security issues concerning individuals or organizations can develop into disastrous scenarios, like losing critical information, promoting relentless attacks (on critical infrastructure as well), or contributing to a distributed denial of service (DDoS) attack [12, 77].

Every single day societies, businesses and citizens in person are under siege by a wide array of cyberthreats including malware, worms, trojan horses, spyware, SQLI, XSS, ransomware [49], and many, many more. In a sense all those malicious instances have simply become part of our daily routine. A short while ago, at the beginning of 2018, a banking malware geared towards android has plundered the wallets of unsuspecting bank app users [4]. The new BankBot strain was altered to such a degree that it was granted passage through the Google Play Store antivirus protection, even though BankBot is a well-known malware. The trojan operated under the guise of what was a benevolent application at a first glance, but once set up on an Android device, it proceeded to appropriate the bank's access credentials.

A very well known case of cross-site scripting (XSS) took place when eBay turned out to be vulnerable to attack [33]. In 2014 JavaScript code was being included in costly item's listings. The user only had to click a malicious, but benign-looking listing to have the script seize control of his browser, get redirected to a site that looked exactly like eBay, and have their credit card credentials stolen. A recent report [51] suggests that the vulnerability has not been fixed after all those years.

Early 2018 was marked by a security violation that touched over 150 million users of a popular fitness app, MyFitnessPal. The media coverage of the breach tried to pass the event as "just another day on the Internet"[43]. With the prevailing

risk of both new and known cybersecurity threats it is very tempting to just nod in agreement with that assertion.

The wide range of cybersecurity violations resulted in spurring an array of different detection methods. Two main trends of research and development emerged, namely signature-based and anomaly-based. Signature-based IDS (Intrusion Detection Systems) operate utilising a storehouse of recognized attacks, while anomaly-based methods form a model of 'normal' traffic and go into alert whenever there has been spotted a divergence from the model [31].

The black-hat society utilises numerous obfuscation techniques to deceive the signature-based detectors. According to a recent analysis, known malevolent software can be made absolutely invisible to contemporary anti-malware applications [11].

The security of any given system comes in a direct proportion to the decisions both the organisations and the individuals make with regards to security requirements and their relative prioritisation.

In this section the author lists a number of recent events that make this kind of endeavour relevant.

The thesis is formulated as follows: "It is possible to design complex machine learning algorithms for effective detection of attacks based on network flows, and to design techniques of improving their credibility and security (detection of adversarial attacks)". The aim of the dissertation is therefore to put forward a modification of a machine learning method and a method of detecting adversarial attacks on machine learning. This causes the work to be split into two major parts. The first part is entitled 'Network Intrusion Detection', and the second part 'Adversarial Attack Detection'.

2. Part One: Network Intrusion Detection

The analysis performed by an IDS is based on local data provided by a set of independent probes deployed in different places of the monitored distributed system. Each of these observation devices is responsible for monitoring a small, well-defined part of the entire system and reporting what is going on. Depending on whether a probe monitors the activity of a particular machine or the activity at a particular point in the communication network, IDSes are classified in three main classes, namely HIDS (Host based Intrusion Detection System), NIDS (Network based Intrusion Detection System), and Hybrid IDS. Whatever the observed targets (computation devices or communication links), the reported information either corresponds to a raw observation of an activity (i.e., an occurrence of an event) or is the result of a first low level analysis that identifies a suspicious local behavior (i.e., a low level alert). The locally generated information is usually stored in logs and journals. It is also sent in messages to an analysis tool that centralizes the data and carries out a deeper global analysis: this activity is a key feature of a Security Information and Event Management (SIEM). Note that this two-level analysis (local and global) can be replaced by a more complex hierarchical structure dedicated to data collection and analysis. In any case, the gathered information is ordered according to the occurence date of each element and thus a unique flow (or stream) of timestamped data (events and low level alerts) is created. While combining outputs from different sources/probes, a pre-processing step is often performed to homogenize, correct, and reduce the flow of elements [74]. Indeed, since the probes are heterogeneous, the information they provide in their messages has to be restructured and standardized using a unique description language.

The lowest common denominator of cyber-attack detection are the machine learning methods: the detection is as weak (or strong), as weak are the data processing approaches.

Two classical approaches are covered and discussed separately even if one of the proposed challenges is to combine them: the signature-based approach and the anomaly-based approach

A bulk of state-of-the-art research does not provide reliable performance results since they rely on either the KDD99 or NSL-KDD benchmark datasets, which is concocted of traffic that is over 20 years old, hence it does not represent

recent attack scenarios and traffic behaviours. Obtaining traffic from simulated environments can help overcome this issue when merged with testing more recent datasets, such as the CICIDS 2017 [7]. Published datasets are available for different domains, such as industrial control systems (ICS) [30]. The observed data that will be provided to us in the context of Sparta will complete the currently limited panel of available datasets.

2.1. The Signature-based Approach

A signature-based approach relies on the apriori knowledge of some possible attacks. Patterns of malicious traffic/activity are compared to current samples, and if a match is found an alarm is raised

The signature-based approach has the advantage of causing very few false positives if the description of the attack and the associated correlation rules are sufficiently accurate. Regarding the false negatives, only the known attacks can be detected. A new attack or even an attack that is intentionally slightly different from a known one are not necessarily detected (0-day exploit). Moreover, the detection of a known attack occurs only if the probes are in sufficient number, well placed to cover all the system and correctly configured.

2.2. The Anomaly-based Approach

The following procedure can serve as an outline for the approach: firstly, the pattern of normality (normal traffic/activity) has to be established and then matched against the current traffic/activity samples. Whenever, the pattern deviates from the established model an alarm is raised. This approach, however, is plagued by false positives (false alarms). Quite frequently, when the characteristics of network traffic (or e.g. HTTP requests in the application layer) evolve, such situation is interpreted as anomalous, even though it is just an intrinsic feature of network usage and of network users behaviour [6].

Concisely put, in setups where new attacks (or even slightly modified families of malware) emerge continuously, the standard protection systems become inconsequential until relevant signatures are collected [21]. On the other hand, anomaly-based approaches (systems which detect abnormalities in traffic, e.g. abnormal requests to databases) tend to produce false positives (false alarms) [2, 63].

Thus numerous machine learning anomaly detection techniques are evaluated by the scientific community.

2.3. Machine Learning for Intrusion Detection - State of The Art

Contemporary Artificial Intelligence-based Intrusion Detection algorithms suffer from two main challenges. The traditional Machine Learning (ML) algorithms are relatively fast, but at the same time encounter a high false positive (FP) rate. On the other hand, Deep Learning displays high accuracy, low FP rates, but a rather cumbersome computational time. Thus, the authors of [84] propose a solution that gives the user the best of both worlds. The proposed solution is an OS based monitoring algorithm that utilises standard ML as a fairly quick monitoring device, called the 'standard stage', and when classification occurs which falls into the 'borderline' status, the second stage of the algorithm is initialised. The second stage, called 'uncertain' utilises Deep Learning as the definitive decision-maker on the maliciousness of a given process.

A Flow-Based Malware detection using a Convolutional Neural Network is evaluated in [81]. The authors suggest that the contemporary detection methods are too dependent on selected packet fields, like the port number, which presents a blind spot for malware using unpredictable port numbers and protocols. Instead, 35 features extracted from packet flows of the data coming from the Stratosphere IPS project are proposed. 2000 datapoints were selected in each class to solve the data balance problem. The authors of [81] used Netmate to extract 35-flow static features to feed to a Convolutional Neural Network and three other ML algorithms, namely a support vector machine (SVM), a random forest (RF), and a multi-layer perceptron (MLP) for evaluation. The models were trained using data from the Stratosphere IPS project (public data). The CNN architecture consisted of one input layer, five feature map layers, one flatten layer, two hidden layers, and one output layer. The authors conclude that the RF algorithm outperforms the remaining methods on all three evaluated indicators - Accuracy, Specificity and Sensitivity. The CNN came in a close second.

In [60] the authors propose a novel, bioinspired way of stegomalware detection, based on an Artificial Immune System (AIS). It is capable of detecting code hidden using three common-use steganographic tools - F5, Outguess and Steghide.

The proposed procedure extracts features from JPEG images with the use of Haar Wavelets. An AIS is a sort of self-defining system, revolving around the creation, identification and maintenance of 'self', and vanquishing of anything that does not meet the definition of 'self'. This particular AIS is based on Negative Selection. The trained system was compared to a method based on SVM and it demonstrated better detection rate and significantly faster performance.

In the case of mobile malware, [80] illustrates that the actions it can employ are of a wide variety. That can make spotting mobile malware an ambitious endeavor. The neverending adaptation to new circumstances causes the prevention

attempts to be an uphill battle. As a novel solution, the authors propose the 'DeepRefiner', a semantic-based deep learning algorithm, displaying an accuracy of 97.74% with a false positive rate of 2.54%, outperforming the comparison, industry-standard methods by a wide margin.

In [78] the problem of selecting proper features for IDS is evaluated. An idea of a hierarchical spatial-temporal features-based intrusion detection system (HAST-IDS) is proposed. The architecture would first recognize the low-level features with the use of a CNN and then the high-level features of traffic with the employment of an LSTM. The feature learning is performed automatically, in its entirety. This process reduces the false alarm rate. Two benchmark datasets are used to test-run the idea: DARPA1998 and ISCX2012. The performance of this set-up is superior in terms of accuracy, detection rate and false positive rate, as compared to other state-of-the-art methods.

In [76] authors extensively test Deep Neural Networks for the use in IDS over a range of old and new datasets. They also come up with a scalable, hybrid, DNN-based model which is suitable for real-time use. An attack could be roughly systematized into five parts: reconnaissance, exploitation, reinforcement, consolidation and finally pillage. When the malicious user reaches the fourth stage the system is basically compromised.

The authors focus on the premise of self-learning systems, with either supervised, semi-supervised or unsupervised algorithms. The main problem with contemporary machine learning solutions is their high false-positive rate and their high computational cost. The authors attribute this fault to learning TCP/IP patterns in a local fashion, as opposed to the Deep Learning solutions. The authors use a variety of benchmark datasets: KDDCup 99, NSL-KDD, Kyoto, UNSW-NB15, WSN-DS, CICIDS2017.

In [62] an evaluation of both shallow and deep neural networks is performed, as per their performance in the domain of intrusion detection. The tests are run over the KDDCup99 dataset. DNN setups from one to five hidden layers, and a suite of classical machine learning algorithms, including AdaBoost, Decision Tree, K-Nearest Neighbour, Linear Regression, Naïve Bayes, Random Forest, SVM-linear and SVM-rbf. The deep setups use a combination of dense and dropout layers one after another to achieve regularization. After comparison, it is clear that a 3-layer DNN outperforms all the other algorithms and setups for the KDDCup99 benchmark dataset.

The authors of [52] perform yet another comparison of deep learning architectures' performance over the NSL-KDD dataset. The methods evaluated are autoencoders (sparse, denoising, contractive, convolutional), LSTM and CNN. Autoencoders learn the latent representation of their inputs but changing the feature space by reducing dimensions. Autoencoders have a 3-layer structure, input layer, hidden layer and an output layer. Input and output layers contain the same number

of nodes. When the hidden layer has fewer neurons than the input/output layers, it is called a bottleneck/discriminative/coding/abstraction layer. Forcing a bottleneck makes the autoencoder acquire the most significant features. A multi-layer setups can also be possible for autoencoders. A sparse autoencoder employs a sparsity penalty and a reconstruction error at the discriminative layer. The prominent characteristic of a sparse autoencoder is that it mitigates overfitting. Denoising autoencoders recover the uncorrupted version of a sample by minimizing the objective function. The unique advantage of Contractive Autoencoders (ContAE) is that they learn representations in a way that is resilient to small changes in data. This characteristic is attained with the imposition of a Forbnius norm of the Jacobian matrix for the encoder activations. The penalty term performs a compression of the localized space, and through this compression, robustness to small changes is gained. Convolutional autoencoders (ConvAE) use a Stochastic Gradient Descent (SGD) to find non-trivial features, which are good initialisations for a CNN, avoiding local minima. A fully connected ConvAE, in contrast to other AE's, preserve the special locality of features.

The methodology used in the evaluation of the DNN setups is as follows: all the architectures, with the exception of ConvAE, are the same. The Autoencoders are prepared with NSL-KDDTrain+. The bottleneck layer reduces the feature space to 16 dimensions from 41. The reduced representations become the input of an MLP. For sparce AE L1 regularisation is performed over the input of the bottleneck. For DAE, the input is treated with Gaussian noise, with level of corruption at 50%. In Contractive AE the loss function is replaced with a contractive loss. ConvAE needs input in the form of an image. NSL-KDD is converted to a 32x32 2d array. This method allows for the discovery of localized relationships in the dataset. The weights of a ConvAE are shared among all locations of the input matrix. The deep setups are compared with classic ML algorithms – Extreme Learning Machine, k-NN, Decision-Tree, Random Forest, SVM, Naïve-Bayes and QDA. DCNN and LSTM models perform at the accuracy of 85% and 89%, beating the remaining setups.

2.4. Artificial Neural Networks in Intrusion Detection

Cybersecurity is a very broad topic, with different measures designed to counter different attack vectors [20]. The application of Artificial Neural Networks (ANN) for intrusion detection systems (IDS) and malware detection is hardly a new concept. There have been evaluations of the notion of using ANN to aid anomaly detection and malware detection as far as in 2009 [64]. In [28] the authors try to address the problems of overfitting, high memory consumption and high overhead of standard IDS / malware detection with a feed-forward ANN. Specifically, a 2-layered feed-forward ANN was recommended. The aforementioned

problems were handled through conjugated training function and validation dataset. The authors claim that their method achieves similar results to classical procedures, but with less computational overhead. The procedure was tested on the benchmark KDD'99 dataset. The conclusion of the paper states that less data is better because of the time the machine needs to crunch it.

In [23] pruning of the ANN is evaluated as part of the optimisation of the network. It is basically the deletion of neural nodes of either the input or the hidden layers. This makes the ANN faster, as less computations have to be processed. In [50] an Artificial Neural Network also showed promise as an IDS when evaluated. The results were, in fact, very encouraging.

In [70] Principal Component Analysis (PCA) is employed as a feature extractor, before feeding the data to the ANN, as opposed to providing the inputs directly from the dataset. As the article illustrates, this drops the memory requirements of the method significantly, along with the time of training necessary. The two evaluated methods displayed comparable results as far as the accuracy is concerned. This makes applying PCA clearly the better option. Using a Kernel PCA betters the training time of ANN, but uses significantly more memory than traditional PCA. Both methods have similar accuracy measures, so the authors of [58] conclude that using a mix of different algorithms is preferable.

There has been research on utilising Graphical Processing Units to accelerate ANN based IDS, since GPU's are a good fit for ANN computations. An increase in performance has been proven [75]. The authors of [71] evaluate an ANN with one hidden layer in comparison with a Support Vector Machine, a Naive Bayes and a C4.5 algorithm. The ANN achieves comparable, or better results in malware detection, but thanks to the simpler nature of a 3-layer ANN framework requires less computations than other tested algorithms. The experiments were performed on the NSL-KDD dataset, which is the current benchmark, and the successor of KDD'99.

2.5. The Proposed Method Based on Artificial Neural Network

Artificial Neural Networks (ANN) are an all-purpose utility for modeling. With a myriad of applications, they are an accepted and renowned tool for data mining, with classification, regression, clustering and time series analysis abilities. The basic assumption of an ANN is that it imitates, to a certain extent, the learning competencies of a biological neural network, stressing by principle the properties of neural networks found in human brains, although strongly streamlined. [48]

The surprising modeling capacity of ANN in pattern recognition derives from its strong malleability as it fits to data. This extensive approximation capacity is markedly important when handling real-world data, when the information is plentiful, but the patterns buried in it remain uncovered.

In an ANN knowledge is gained through updating weights with consecutive batches of data instances. The algorithm can recognise the associations among the variables, as well as generalise in a way that allows for high performance on new, unforseen data. [22] It is basically like fitting a line, or a plane, or a hyper-plane through a set. [57].

An artificial neural network with a sole computational layer is dubbed a perceptron. It consists of an input and an output, computational layer. After the data points are fed to the input layer, they are issued to the computational layer. The input layer contains d nodes that speak for d features $X = [x_1...x_d]$ and edges of weight $W = [w_1...w_d]$. The output neuron computes $W \cdot X = \sum_{i=1}^{d}(w_i x_i)$. In case of the perceptron, the forecast is binary, and is delineated by the sign of the value that is the result of the output layer computation. To help deal with distribution imbalance, bias can be added.

The prediction of $\hat{y}$ is the result of the following equation:

$$\hat{y} = sign\{W \cdot X + b\} = sign\{\sum_{i=1}^{d} w_i x_i + b\}$$

As seen in the equation, the $sign$ is the activation function $\Phi(v)$. Numerous activation functions can be utilised in artificial neural networks with multiple hidden layers. For ease of training it is commonly either the Rectified Linear Unit (ReLU) or Hard Tanh in multilayered networks. The error of the regression can be indicated as the difference between the real-life test value and the predicted value, so $E(X) = y - \hat{y}$. If the error is not equal to 0 the weights should be amended. Thus, the purpose of the perceptron is to minimise the least-squares between y and $\hat{y}$, for all data points in dataset D. This objective is dubbed the loss function.

$$\sum_{(X,y)\in D} (y - sign\{W \cdot X\})$$

The loss function is defined over the whole dataset X, the weights W are updated with the learning rate α, and the algorithm iterates over the entire dataset until it converges. This algorithm was named stochastic gradient-descent, also expressed by:

$$W \Leftarrow W + \alpha E(X)X$$

[1]

A multi-layer neural network is created via multiple computational layers, also named the hidden layers. The title itself hints to the black-box character of those layers, as the computations are shrouded from the user's perspective. The data points are carried from the input layer subsequent layers with computations at every stage, down to the output layer.

The aforementioned procedure is referred to as the feed-forward neural network [1]. The exact count of nodes in the foremost computational layer usually does not reach the count of nodes of the input layer. The particular number of neurons and the number of hidden layers is in proportion to the intricacy of the necessary model and, of course, on the data[22]. While in some special cases utilising a fully-connected layer is the norm, the use of hidden layers with the count of neurons below that of the inputs grants a loss in representation, which oftentimes increases the network's performance. This is very likely as a result of getting rid of the noise in data [1].

A network built with too many neurons can display unwanted behaviour known as overfitting. Also named overtraining, this particular phenomenon happens when the artificial neural network fitted the exact patterns found in the training dataset so tightly that it has trouble performing on unforseen data, as the approximation is not generalised enough. [1]

In parts of this work the influence of the number of hidden layers as well as the number of neurons in those hidden layers on the ANN performance has been subjected to scrutiny. While it can be considered as architecture, or network structure, those aspects could also be considered hyperparameters.

2.5.1. The Usage of Backpropagation

Having to train a single-layer perceptron is quite simple - the loss function is simply a function of the weights. With multiple layers the procedure gets messy as it makes many layers of weights influence one another. Backpropagation calculates the Error Gradient as the sum of local-gradients over multiple paths to the output node [1]. The algorithm consists of two phases - the forward and the backward phase. In the forward phase the data points are served to the input nodes, and one after one the results at consecutive layers are computed with the current weights. The result of this prediction is compared to the training instance. The backward phase uncovers the gradient of the loss function for all the weights. The gradients update the weights, starting from the output layer, stepping back all the way to the first layer. This weight updating process iterates over the training data - each iteration is called an epoch - ANN's can often take thousands of those iterations to attain convergence.

2.5.2. Improving the selected algorithms with hyperparameter optimization

A most important part of the Artificial Neural Network design comes in the role of the activation function, as the effect it carries over the achievable results is straightforward.

One can have diverse types of activation functions. The decision on the type of an activation function plays a crucial role especially in multi-layer networks,

as each layer can have its own non-linear activation function[1]. Each distinctive function can have a special influence on the results of the ANN, as well as how the ANN converges, and the comprehensive nature of the network. Out of a wide range of activation functions $\Phi(v)$ four were selected:

- Sigmoid
- Hard Sigmoid
- Rectified Linear Unit (ReLU)
- Hyperbolic Tangent (tanh)

The optimal network setup is found by using a grid search procedure, which completes an all-encompassing search over the hyperparameter's space. The grid search parameters included:

- the epoch count
- the batch size
- the activation function
- the optimiser
- in some tests the number of hidden layers
- in some tests the number of neuron nodes

The full cycle of learning and adapting the weights of the network is called an epoch. The particular count of samples utilised in one iteration is called batch size [1]. The grid search can test different activation functions and optimisers. The optimisers evaluated in this paper are:

- Adaptive Moment Estimation (adam)
- Root Mean Square Propogation (rmsprop)
- Stochastic gradient descent (SGD)

2.5.3. Dimensionality Reduction

The search for a feature vector which communicates the nature of the dataset, but without having to represent every single feature is named dimensionality reduction.

It is about the construction of an n-dimensional projection that explains the data of a k-dimensional space. Computational benefits of this procedure are the first ones that come to mind, and these are closely followed by preventing the 'curse of dimensionality'. Said curse makes ML classifiers fall short of the expected results with the increase of dimensions [48]. Exponential increase of samples is necessary for the algorithms to be back on track.

A wide range of approaches to dimensionality reduction exist. Since negligible features are basically just noise, frequently, feature selection is implemented

to acquire the most applicable feature set. There are, however, other methods to arrive at a smaller feature set which explains the data thoroughly.

Another of the go-to methods of dimensionality reduction is Principal Component Analysis (PCA). To put it shortly, PCA looks for a view of the data where the variance is maximised. An example introduced in [48] shows that if the data creates a line, performing PCA would immediately indicate that the variance over but one dimensions equals 0. Thus, since the features of those dimensions are useless, they can be gotten rid of. Even though data-gathering could suggest a high signal strength in one direction, the data will most likely contain noise in a lot of features. Provided the signal overpowers the noise to a high enough extent, the elicited projection containing maximum variance is highly likely to convey the essence of the data.

2.6. Experimental Setup and Results

2.6.1. Description of chosen datasets

NSL-KDD

NSL-KDD is a data set created to address the problems of the KDD'99 malware and intrusion data, which were repeatedly raised in the literature. It is now an established benchmark dataset, even though it still displays some of the unwanted characteristics. Still, the lack of open IDS datasets and the difficulty in collecting the data makes NSL-KDD the go-to solution for intrusion detection / malware detection research.

The set contains almost 5.000.000 records, which makes it both suitable for machine learning, and not overbearingly humongous so as to force researches to pick parts of the set randomly. This makes the results more easily comparable.

The NSL-KDD is cleared of redundant data, to prevent ML algorithms bias, which is an improvement over the original KDD'99 dataset.

Intrusion Detection Evaluation Dataset - CICIDS2017

CICIDS2017 [66] was created as a response to the lack of dependable and adequately recent cybersecurity datasets. The IDS datasets available to researchers usually come with a set of their own problems - be it lack of traffic diversity, lack of attack variety, insufficient features and so on. The authors of CICIDS2017 offer a dataset with realistic background traffic, created by abstracting the behaviour of 25 users across a range of protocols. The data was captured over a range of 5 days, with 4 days the setup being assaulted by a range of attacks including a range of malware, DoS attacks, web attacks and others. This paper utilises the captures from Tuesday - with FTP-Patator and SSH-Patator attacks. Not only

Tablica 1: CICIDS2017 initial results

	precision	recall	f1-score	support
0	0.99	1.00	1.00	43171
1	1.00	0.98	0.99	820
2	1.00	0.54	0.70	574
micro avg	0.99	0.99	0.99	44565
macro avg	1.00	0.84	0.89	44565
weighted avg	0.99	0.99	0.99	44565
samples avg	0.99	0.99	0.99	44565

is CICIDS2017 one of the newest datasets available to researchers, but it also features over 80 network flow features.

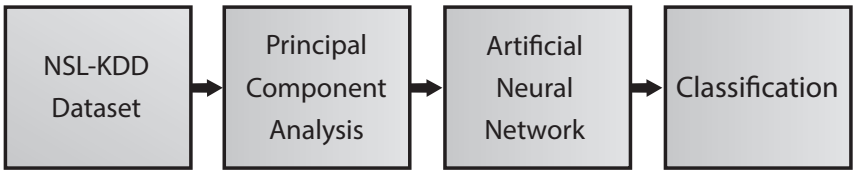
2.6.2. Experimental Setup

The experiments were set up on the same 7th generation Intel Core i7-7500U CPU with two cores of 2.7 and 2.9 GHz and 16GB RAM. The testing environment was set up using the Keras interface for the TensorFlow library running on Python 3.5. Multiple architecture setups were tested, ranging from 1 hidden layer of 25 neurons, which constitute a half of the count of neurons in the input layer, to a network of 4 layers 25 neurons each. PCA is then performed to get the number of features down to 50. The number of extracted features was an arbitrary decision based on initial tests of the setup. This reduced feature vector constitutes the feature-set fed to the input layer of the Artificial Neural Network. The pipeline of the process is illustrated in Fig.1

In a new batch of experiments the algorithm has been applied to the CICIDS2017 dataset. The initial results found in Tab.1 were very encouraging, with accuracy exceeding 99% (0.9936). The recall of 0.54 and f1-score of 0.70 in one of the classes signified that there might be a balancing problem in the dataset. A closer inspection revealed that there are over 43000 records of benign netflows in the set, but only slightly over 1300 attack records. This is shown in Tab. 1 in the 'Support' column. To counteract that the majority class was randomly subsampled with the number of samples matching the sum of attack records. The initial results for the balanced dataset are represented in Tab.2. The accuracy of the procedure on the balanced dataset exceeded 97% over the test set, and 95% mean accuracy in the 10-fold cross-validation. The results with other methods of dataset balancing, like the one shown in [35] or SMOTE [17] are left for future work.

Tablica 2: Balanced CICIDS2017 initial results

	precision	recall	f1-score	support
0	0.99	0.96	0.98	1387
1	1.00	1.00	1.00	804
2	0.92	0.99	0.95	576
micro avg	0.98	0.98	0.98	2767
macro avg	0.97	0.98	0.98	2767
weighted avg	0.98	0.98	0.98	2767



Rysunek 1: The process pipeline starting with the NSL-KDD dataset. Similar process is used for CICIDS2017.

2.6.3. Cross-Validation

Machine Learning and numerous facets of Artificial Intelligence come with their own set of problems. Some of those problems come in the form of selection bias, or overfitting. If the algorithms experience those challenges they might perform outstandingly in the lab, but when subjected to real data their performance will not be satisfactory. In order to mitigate this phenomenon the models undergo a procedure called k-fold cross validation, also known as rotation estimation. The training dataset is split into k parts, and k-1 parts are used for training, with the remaining one is utilized for testing. Then the procedure is repeated k times changing the testing part. The folds are sampled randomly [34] [32]. In this work 10-fold cross-validation was used.

2.6.4. Comparison to other state of the art machine learning algorithms

In this section the performance of other ML approaches is presented. To place the performance of the illustrated approach in context, tests were performed using a Support Vector Machine with gaussian kernel(SVM), the Naïve Bayes classifier and ADABoost. Table 4 illustrates the differences among the results the classifiers have achieved with the use of CICIDS2017 data.

Tablica 3: 1 layer 10 neurons over the NSL-KDD dataset

Accuracy	Optimizer	Epochs	Activation	Batch Size
0.998915	adam	300	sigmoid	10
0.998516	rmsprop	300	relu	10
0.998761	adam	300	relu	10
0.997432	SGD	300	relu	10
0.998775	rmsprop	300	relu	100
0.998881	adam	300	relu	100
0.995214	SGD	300	relu	100
0.998450	rmsprop	300	sigmoid	10
0.998915	adam	300	sigmoid	10
0.995691	SGD	300	sigmoid	10
0.998709	rmsprop	300	sigmoid	100
0.998717	adam	300	sigmoid	100
0.990996	SGD	300	sigmoid	100
0.998562	rmsprop	300	tanh	10
0.998829	adam	300	tanh	10
0.996776	SGD	300	tanh	10
0.998765	rmsprop	300	tanh	100
0.998771	adam	300	tanh	100
0.994085	SGD	300	tanh	100
0.998522	rmsprop	300	hard_sigmoid	10
0.998833	adam	300	hard_sigmoid	10
0.995639	SGD	300	hard_sigmoid	10
0.998683	rmsprop	300	hard_sigmoid	100
0.998845	adam	300	hard_sigmoid	100
0.991385	SGD	300	hard_sigmoid	100

Tablica 4: The results of other ML methods over the CICIDS2017 dataset

	precision	recall	f1-score	support
SVM , Accuracy: 0.9584				
0	1.00	0.92	0.96	1387
1	0.97	1.00	0.98	804
2	0.87	0.99	0.93	576
micro avg	0.96	0.96	0.96	2767
macro avg	0.95	0.97	0.96	2767
weighted avg	0.96	0.96	0.96	2767
Naive Bayes , Accuracy: 0.9078				
0	1.00	0.82	0.90	1387
1	0.86	1.00	0.93	804
2	0.82	1.00	0.90	576
micro avg	0.91	0.91	0.91	2767
macro avg	0.89	0.94	0.91	2767
weighted avg	0.92	0.91	0.91	2767
ADABoost , Accuracy: 0.9382				
0	1.00	0.88	0.93	1387
1	1.00	1.00	1.00	804
2	0.78	0.99	0.87	576
micro avg	0.94	0.94	0.94	2767
macro avg	0.92	0.96	0.93	2767
weighted avg	0.95	0.94	0.94	2767

Tablica 5: 4 hidden layers, 25 neurons each, CICIDS2017 dataset

Accuracy	Activation	Batch Size	Optimizer	Epochs
0.975176	relu	50	adam	30
0.976983	relu	50	rmsprop	30
0.970597	relu	50	SGD	30
0.974252	relu	100	adam	30
0.976742	relu	100	rmsprop	30
0.958184	relu	100	SGD	30
0.970516	sigmoid	50	adam	30
0.969793	sigmoid	50	rmsprop	30
0.930227	sigmoid	50	SGD	30
0.969110	sigmoid	100	adam	30
0.965575	sigmoid	100	rmsprop	30
0.585017	sigmoid	100	SGD	30
0.972444	hard_sigmoid	50	adam	30
0.973408	hard_sigmoid	50	rmsprop	30
0.708456	hard_sigmoid	50	SGD	30
0.971922	hard_sigmoid	100	adam	30
0.972645	hard_sigmoid	100	rmsprop	30
0.499819	hard_sigmoid	100	SGD	30

2.7. Results

Hyperparamater optimisation is performed on each of the setups. The grid-search method evaluates each of possible permutations of the selected hyperparameters. Namely, the used epochs count, the batch size, the optimiser and the activation function are consecutively permutated in order to achieve the highest accuracy. The tables below illustrate the way the accuracy fluctuates on various ANN setups. The optimal setup for the algorithm in the CICIDS2017 case has been established with the gridsearch procedure as well. A sample of the results can bee seen at Tab. 5. It is apparent that the results acquired with different parameter setups vary greatly, just as it was in the NSL-KDD case.

2.8. Machine Learning Approach Enhanced with Data Balancer

The focus of this research lies on the impact the balance of the instance numbers among classes in a dataset has on the performance of ML-based classification methods. In general, the step-by-step process of ML-based Intrusion Detection System (IDS) can be succinctly summarised as follows: a batch of annotated data

Rysunek 2: IDS training pipeline with dataset balancing

is used to train a classifier. The algorithm 'fits' to the training data, creating a model. This is followed by testing the performance of the acquired model on the testing set - a batch of unforeseen data. In order to alleviate the data balancing problem present in the utilised IDS dataset an additional step is undertaken before the algorithm is trained (as seen in Fig. 2).

The ML-based classifier block of Fig. 2 can be realised by an abundance of different machine learning methods. In fact, recent research showcases numerous novel approaches including deep learning [53][54], ensemble learning [83][38], various augmentations to classical ML algorithms [39] etc. In this work three basic models were chosen to put emphasis on the data balancing part. These are:

- Artificial Neural Network [68][70]
- Random Forest [8]
- Naive Bayes [48]

These represent three significantly different approaches to machine learning and were selected to cover possibly the widest range of effects dataset balancing could have on the effectiveness of ML.

2.9. Balancing Methods

In the cases suffering from the data imbalance problem the number of training samples belonging to some classes is larger in contrast to other classes.

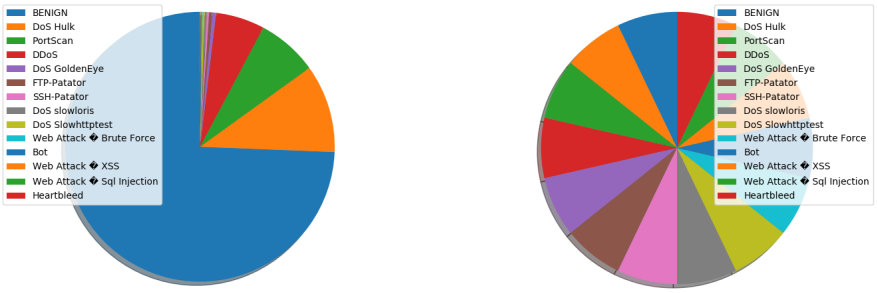
The conundrum of data imbalance has recently been deeply studied in the area of machine learning and data mining. In numerous cases, this predicament impacts the machine learning algorithms and in result deteriorates the effectiveness of the classifier [36]. Typically in such cases, classifiers will achieve higher predictive accuracy over the majority class, but poorer predictive accuracy over the minority class. In general, solutions to this problem can be categorised as (i) data-related, and (ii) algorithm-related.

Tablica 6: CICIDS2017 (pełen zbiór) / Niezbalansowany

	ANN ACC: 0.9833			RandomForest ACC: 0.9987			NaïveBayes ACC: 0.2905			
	precision	recall	f1-score	precision	recall	f1-score	precision	recall	f1-score	support
0	0.99	0.99	0.99	1.00	1.00	1.00	1.00	0.10	0.18	162154
1	0.97	0.35	0.52	0.88	**0.68**	0.77	0.01	0.65	0.01	196
2	1.00	1.00	1.00	1.00	1.00	1.00	0.94	0.95	0.94	12803
3	0.99	0.97	0.98	1.00	0.99	1.00	0.09	0.93	0.16	1029
4	0.95	0.94	0.94	1.00	1.00	1.00	0.74	0.70	0.72	23012
5	0.89	0.98	0.93	0.96	0.98	0.97	0.00	0.67	0.01	550
6	0.99	0.98	0.99	1.00	0.99	0.99	0.05	0.52	0.09	580
7	0.99	0.98	0.99	1.00	1.00	1.00	0.10	0.99	0.18	794
8	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1
9	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.99	0.99	15880
10	1.00	0.49	0.66	1.00	1.00	1.00	0.08	0.99	0.15	590
11	0.85	0.10	0.17	0.86	**0.99**	0.92	0.00	0.07	0.00	301
12	0.00	0.00	0.00	1.00	1.00	1.00	0.01	1.00	0.02	4
13	1.00	0.02	0.05	0.95	**0.61**	0.74	0.08	0.93	0.14	130
macro avg	0.90	0.70	0.73	0.97	0.95	0.96	0.36	0.75	0.33	218024
weighted avg	0.98	0.98	0.98	1.00	1.00	1.00	0.95	0.29	0.34	218024

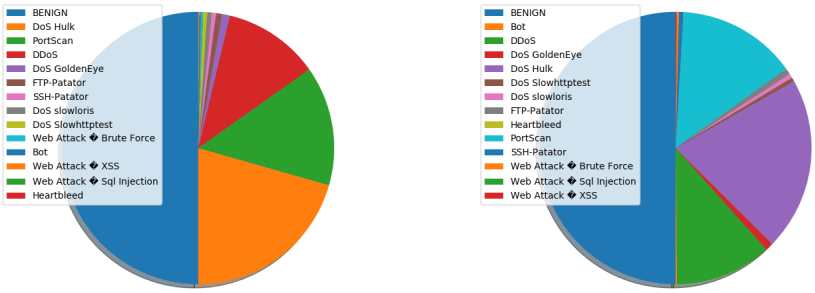
Tablica 7: CICIDS2017 / Random Subsampling

	ANN ACC: 0.9812			RandomForest ACC: **0.9980**			NaïveBayes ACC: 0.2911			
	precision	recall	f1-score	precision	recall	f1-score	precision	recall	f1-score	support
0	1.00	0.98	0.99	1.00	1.00	1.00	1.00	0.10	0.18	162154
1	0.50	0.63	0.56	0.91	**0.92**	0.91	0.01	0.65	0.01	196
2	1.00	1.00	1.00	1.00	1.00	1.00	0.94	0.95	0.94	12803
3	0.98	0.98	0.98	1.00	1.00	1.00	0.09	0.93	0.16	1029
4	0.90	0.99	0.95	1.00	1.00	1.00	0.74	0.70	0.72	23012
5	0.90	0.99	0.94	0.98	0.99	0.99	0.00	0.67	0.01	550
6	0.97	0.98	0.97	0.99	0.99	0.99	0.05	0.52	0.09	580
7	0.99	0.98	0.98	1.00	1.00	1.00	0.10	0.99	0.19	794
8	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1
9	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.99	0.99	15880
10	0.97	0.49	0.65	1.00	0.99	1.00	0.08	0.99	0.15	590
11	0.59	0.23	0.33	0.80	**0.97**	0.88	0.00	0.07	0.00	301
12	0.00	0.00	0.00	1.00	0.80	0.89	0.01	1.00	0.02	4
13	0.80	0.03	0.06	0.96	**0.40**	0.57	0.08	0.93	0.15	130
macro avg	0.83	0.73	0.74	0.97	0.93	0.94	0.36	0.75	0.33	218024
weighted avg	0.99	0.98	0.98	1.00	1.00	1.00	0.95	0.29	0.34	218024



(a) Original Distribution (b) After SMOTE

Rysunek 1: Class distribution in CICIDS 2017 - Original unbalanced distribution and after SMOTE



(a) BENIGN class randomly subsampled (b) NearMiss

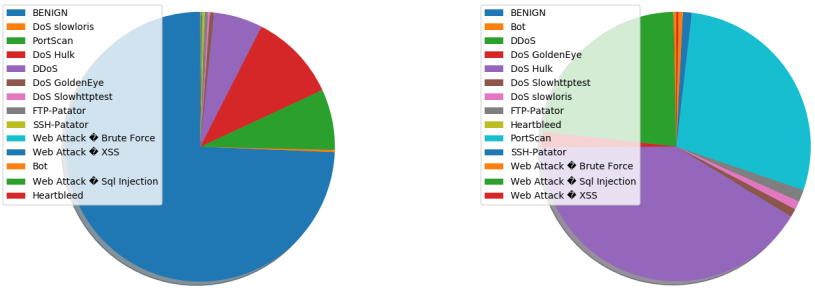
Rysunek 2: Class distribution in CICIDS 2017 - After performing random under-sampling and NearMiss

In the following paragraphs, these two categories of balancing methods will be briefly introduced. The focus of the analysis was on the practical cybersecurity-related application that faces the data imbalance problem.

2.9.1. Data-related Balancing Methods

Two techniques, belonging to this category, that are commonly used to cope with imbalanced data use the principle of acquiring a new dataset out of the existing one. This is realised with data sampling approaches. There are two widely recognised approaches called data over-sampling and under-sampling.

A number of dataset balancing approaches was put to evaluation, among them random subsampling, eliminating tomek links, near-miss, clustering and Borderline synthetic minority oversampling technique (Borderline SMOTE).



(a) After cleaning the Tomek-links (b) Cluster-Centers undersampling

Rysunek 3: Class distribution in CICIDS 2017 - After cleaning the Tomek-Links and performing ClusterCenters undersampling

2.9.2. Algorithm-related Balancing Methods

Utilizing unsuitable evaluation metrics for the classifier trained with the imbalanced data can lead to wrong conclusions about the classifier's effectiveness. As the majority of machine learning algorithms do not operate very well with imbalanced datasets, the commonly observed scenario would be the classifier totally ignoring the minority class. This happens because the classifier is not sufficiently penalized for the misclassification of the data samples belonging to the minority class. This is why the algorithm-related methods have been introduced as a part of the modification to the training procedures. One technique is to use other performance metrics. The alternative evaluation metrics that are suitable for imbalanced data are:

– precission - indicating the percentage of relevant data samples that have been collected by the classifier
– recall (or sensitivity)- indicating the total percentage of all relevant instances that have been detected.
– f1-score - computed as the harmonic mean of precision and recall.

Another technique that is successfully used in the field is a cost-sensitive classification. Recently this learning procedure has been reported to be an effective solution to class-imbalance in the large-scale settings. Without losing the generality, let us define the cost-sensitive training process as the following optimisation formula:

$$\hat{\theta} = \min_{\theta} \left\{ \frac{1}{2}||\theta||^2 + \frac{1}{2} \sum_{i=1}^{N} C_i ||e_i||^2 \right\} \tag{1}$$

where θ indicates the classifier parameters, e_i the error in the classifier response for the i-th (out of N) data samples, and C_i the importance of the i-th data sample.

In cost-sensitive learning, the idea is to give a higher importance C_i to the minority class, so that the bias towards the majority class is reduced. In other words, we are producing a cost function that is penalizing the incorrect classification of the minority class more than incorrect classifications of the majority class.

In this paper we have focused on **Cost-Sensitise Random Forest** as an example of cost-sensitive meta-learning. This is mainly due to the fact the Random Forest classifier in that configuration yields the most promising results. These can be found in Tab. 10

2.9.3. Results and Perspectives

CICIDS 2017 dataset consists of 13 classes - 12 attacks and 1 benign class. As depicted in Fig. 1, there is a wide discrepancy among the classes in terms of the number of instances, especially the benign class as compared to the attack classes.

During the tests the initial hypothesis was that balancing the classes would improve the overall results. Random Subsampling (Tab. 7) along a slew of other subsampling methods were used to observe the influence dataset balancing has on the performance of 3 reference ML algorithms - an Artificial Neural Network (ANN), a RandomForest algorithm and a Naive Bayes classifier. Finally, Borderline SMOTE was conducted as a reference oversampling method. Balancing the benign class to match the number of samples of all the attacks combined changed both the precision and the recall achieved by the algorithm. It also became apparent that none of the subsampling approaches outperformed simple random subsampling in the case of CICIDS2017. The tests revealed an interesting connection among the precision, recall and the imbalance ratio of the dataset. Essentially, there seems to exist a tradeoff between precision and recall that can be controlled by the number of the instances of classes in the training dataset. To evaluate that assertion further tests were conducted. Random Forest algorithm was trained on the Unbalanced dataset and then all the classess were subsampled to match the number of samples in one of the minority classes.

The tests proved that changing the balance ratio undersampling the majority classes improves the recall of the minority classes, but degrades the precission of the classifier on those classes. This basically means that dataset balancing causes the ML algorithms to misclassify the (previously) majority classes as instances of the minority classes, thus boosting the false positives.

Finally, a cost-sensitive random forest algorithm was tested. After trying different weight setups results exceeding any previous undersampling or oversampling methods were attained (Tab. 10). It is noteworthy that the achieved recall for class 13 is higher while still retaining a relatively high precision. A relationship

Tablica 8: CICIDS2017 / Random Subsampling down to 7141 instances per class / RandomForest

	precision	recall	f1-score	support
0	1.00	0.98	0.99	162154
1	0.13	0.99	0.23	196
2	1.00	1.00	1.00	12803
3	0.92	1.00	0.96	1029
4	0.98	1.00	0.99	23012
5	0.85	0.99	0.92	550
6	0.93	0.99	0.96	580
7	0.93	1.00	0.96	794
8	0.17	1.00	0.29	1
9	1.00	1.00	1.00	15880
10	0.73	1.00	0.85	590
11	0.63	0.98	0.77	301
12	0.07	1.00	0.14	4
13	0.32	0.48	0.39	130
accuracy			0.9872	218024
macro avg	0.69	0.96	0.74	218024
weighted avg	0.99	0.99	0.99	218024

Tablica 9: CICIDS2017 / Random Subsampling down to 1174 instances per class / RandomForest

	precision	recall	f1-score	support
0	1.00	0.96	0.98	162154
1	0.07	1.00	0.13	196
2	0.99	1.00	1.00	12803
3	0.69	1.00	0.82	1029
4	0.94	0.99	0.97	23012
5	0.76	0.99	0.86	550
6	0.86	0.99	0.92	580
7	0.81	1.00	0.89	794
8	0.17	1.00	0.29	1
9	1.00	1.00	1.00	15880
10	0.44	1.00	0.61	590
11	0.23	0.65	0.34	301
12	0.07	1.00	0.13	4
13	0.13	0.95	0.23	130
accuracy			0.9657	218024
macro avg	0.58	0.97	0.65	218024
weighted avg	0.99	0.97	0.97	218024

Tablica 10: CICIDS2017 / Cost-Sensitive RandomForest

	precision	recall	f1-score	support
0	1.00	1.00	1.00	162154
1	0.34	0.91	0.50	196
2	1.00	1.00	1.00	12803
3	1.00	0.99	0.99	1029
4	1.00	1.00	1.00	23012
5	0.97	0.98	0.97	550
6	1.00	0.99	0.99	580
7	1.00	1.00	1.00	794
8	1.00	1.00	1.00	1
9	1.00	1.00	1.00	15880
10	1.00	1.00	1.00	590
11	0.98	0.85	0.91	301
12	1.00	1.00	1.00	4
13	0.72	0.96	0.83	130
accuracy			0.9973	218024
macro avg	0.93	0.98	0.94	218024
weighted avg	1.00	1.00	1.00	218024

between class 11 and class 13 was also discovered, where setting a higher weight for class 13 would result in misclassification of class 11 samples as class 13 samples and the other way round.

2.10. Gated Recurrent Unit in Network Intrusion Detection

To the best of our knowledge, the Gated Recurrent Unit has not been thoroughly researched for intrusion detection since its inception in 2014 [19]. [41] Evaluates the way Principal Component Analysis improves the results of GRU for Intrusion Detection Systems, achieving, as the authors report, remarkable results.

The use of Variant Gated Recurrent Units (E-GRU) is evaluated in [29] as a preprocessing step in payload aware IDS. The authors notice an improvement over the state-of-the-art methods on a benchmark dataset - ISCX2012. The achieved accuracy reaches 99.9%, however, the memory usage of E-GRU turns out to be 32 times the memory usage of a standard GRU.

The authors of [79] improve the performance of IDS by utilising a Deep Network model with automatic feature extraction. The GRU is used as a feature extractor, which feeds the outputs to a Multi Layer Perceptron (MLP), which then uses a softmax to come up with the final classification. The experiments were

performed on KDD99 and NSL-KDD datasets, achieving 99.98% for KDD99 and 99.55% accuracy for NSL-KDD.

2.11. Recurrent Neural Networks

The usual architectures of neural networks lack the ability to recognise sequential dependencies among data. This makes them underperform on data types that progressively build on previous instances, like time-series data, or text. [9] A different architecture is necessary to handle this kind of data. The answer to this challenge is the Recurrent Neural Network (RNN) - an architecture that features feedback loops. The RNN is capable of learning without relying on the Markov assumption, the premise that, given a present state, all the following states do not depend on the past states [69]. The output from the recent time index T is applied as one of the outputs to time index T+1, or back to itself [26]

Because of this propagation of weights through time the RNN faces its own kind of problem. The mentioned weights are multiplied recursively, thus, if the weights are too small, the subsequent values will be progressively getting smaller and smaller, or, should the weights be sizeable, the final values will approach infinity. This challenge is referred to as the Vanishing Gradient- or the Exploding Gradient problem, respectively [26]

2.11.1. Long Short-Term Memory and GRU

The vanishing/exploding gradient problem arises whenever the matrices are multiplied repeatedly, destabilising the result. This occurs whenever the RNN is dealing with a long enough sequence. For short sequences the RNN never even experiences the problem. Thus, one could stipulate that an RNN has a reasonable short-term memory, but underperforms when long term memory is necessary. To address this problem, the Long Short-Term Memory network was introduced. It uses a property called 'cell state' to retain portions of its long-term memory.

The Gated Recurrent Unit (GRU) [19], can be seen as a simplification of the LSTM architecture, following the same basic principle, but not completely mapping one another. The GRU uses a 'reset gate' to partially reset the hidden states of the network [9]. The models attained with the use of a GRU usually have lesser complexity than those of LSTM [26]. The GRU is more efficient than LSTM [9].

The results are encouraging but clearly, need further research. The architecture we have evaluated achieved a weighted average of 97% across the board, getting up to 100% recognition of some of the attack classes.

On the other hand, some of the attacks were herded with the other classes. These are, as expected, the underrepresented classes, like '1', '3' and '4'. Since

Tablica 11: Random Subsampling-Balanced CICIDS2017, GRU feature extractor, RandomForest classifier accuracy: 0.9918

	precision	recall	f1-score	support
BENIGN	0.99	0.99	0.99	166922
Bot	0.82	0.87	0.84	558
DDoS	0.99	0.99	0.99	38433
DoS GoldenEye	0.87	0.95	0.91	2843
DoS Hulk	0.99	0.99	0.99	69356
DoS Slowhttptest	0.98	0.97	0.98	1666
DoS slowloris	0.98	0.99	0.99	1725
FTP-Patator	1.00	1.00	1.00	2377
Heartbleed	0.67	1.00	0.80	2
PortScan	1.00	1.00	1.00	47633
SSH-Patator	0.98	0.98	0.98	1767
Brute Force	0.67	0.63	0.65	480
Sql Injection	0.00	0.00	0.00	0
XSS	0.29	0.38	0.33	150
macro avg	0.80	0.84	0.82	333912
weighted avg	0.99	0.99	0.99	333912

the test dataset was randomly sampled at the time of the train/test split, not all of the attack types made it into the evaluation.

2.12. Using Gated Recurrent Units for feature extraction

In a final batch of experiments all the preceding improvement methods come together to form a new approach to IDS. Taking the results of previous experiments, the GRU is used as a feature extractor, the dataset is ballanced - either with random subsampling, or through cost-sensitive learning, the influence of PCA is tested and finally, RandomForest is used for classification.

Tablica 12: Random Subsampling-Balanced CICIDS2017, GRU feature extractor, PCA before classifier, RandomForest classifier , accuracy: 0.9927

precision	recall	f1-score	support	
BENIGN	0.99	0.99	0.99	166944
Bot	0.84	0.87	0.86	569
DDoS	0.99	0.99	0.99	38426
DoS GoldenEye	0.89	0.95	0.92	2903
DoS Hulk	0.99	0.99	0.99	69249
DoS Slowhttptest	0.99	0.97	0.98	1676
DoS slowloris	0.98	0.99	0.98	1723
FTP-Patator	1.00	1.00	1.00	2378
Heartbleed	1.00	1.00	1.00	3
PortScan	1.00	1.00	1.00	47630
SSH-Patator	0.98	0.98	0.98	1778
Brute Force	0.66	0.63	0.64	476
Sql Injection	0.00	0.00	0.00	2
XSS	0.24	0.31	0.27	155
macro avg	0.83	0.83	0.83	333912
weighted avg	0.99	0.99	0.99	333912

Tablica 13: Random Subsampling-Balanced CICIDS2017, GRU feature extractor, cost-sensitive RandomForest classifier , accuracy: 0.9903

	precision	recall	f1-score	support
BENIGN	0.99	0.99	0.99	435436
Bot	0.80	0.76	0.78	618
DDoS	0.97	0.97	0.97	38612
DoS GoldenEye	0.79	0.91	0.85	2681
DoS Hulk	0.99	0.99	0.99	69278
DoS Slowhttptest	0.96	0.90	0.93	1758
DoS slowloris	0.93	0.99	0.96	1637
FTP-Patator	0.99	0.99	0.99	2389
Heartbleed	0.67	1.00	0.80	2
PortScan	1.00	1.00	1.00	47636
SSH-Patator	0.98	1.00	0.99	1743
Brute Force	0.73	0.30	0.43	1088
Sql Injection	0.00	0.00	0.00	0
XSS	0.11	0.17	0.13	122
macro avg	0.78	0.78	0.77	603000
weighted avg	0.99	0.99	0.99	603000

3. Part two: Adversarial Attack Detection

3.1. Introduction

Recent advances in machine learning (ML) and the surge in computational power have opened the way to the proliferation of Artificial Intelligence (AI) in all walks of life. The insights inferred from gathered data with the use of ML techniques have radically transformed, for example, the fields of health care and finance, along with uses in security systems [55].

Machine Learning has the uncanny ability to gather the interdependencies among features in large datasets. A range of methods emerged, like the Support Vector Machines (SVM), Clustering, Neural Networks, etc. The ML procedure usually follows the same outline - one where a training phase is followed by a deployment phase, where classification or regression is performed.

The training phase is where the algorithm 'fits' a model to the provided data - usually a large set. ML often achieves surprisingly accurate results in a wide range of applications [3].

With the real-world applications of AI came the realization that its security requires immediate attention. Malicious users, called 'Adversaries' in the AI world, can skillfully influence the inputs fed to the AI algorithms in a way that changes the classification or regression results [15]. Regardless of the machine learning's ubiquity, the awareness of the security threats and ML susceptibility to adversarial attacks is fairly uncommon [55]. Currently, numerous vulnerabilities have been exposed by researchers, most notably speech recognition, autonomous vehicles, and overall deep learning [15].

The research on securing machine learning is ongoing, with a range of different approaches found in recent literature. The puzzle of a truly immune system, however, still remains unsolved. The solutions already developed are yet to be proven in real-life applications as well [45]. The existing fixes introduced contemporary research include approaches like training the algorithms with the inclusion of perturbed examples, concepts like distillation or using a Generative Adversarial Network. They are reported to work for particular types of attacks, though they do not provide security against all types of strikes. Those solutions can also lead to underperforming ML solutions [15].

3.2. Overview of adversarial attacks

Concisely put, exploratory attacks are a way to form a functional equivalent of a deployed ML algorithm, to extract the decision boundary, the setup of the algorithm, its properties and the information about the training dataset [82].

The danger an adversary poses is determined by the information available to it. The level of access the malevolent user has determined the range of attacks they can employ. This is referenced to as Adversarial Capabilities [55, 15]. The level of acquaintance the adversary possesses of the machine learning algorithm architecture influences their behaviour. That level can be broadly categorized as white- and black box.

Black box attacks are undertaken with no prior familiarity with the model or any of its parts. This can be performed by carefully providing and observing the input/output pairs of the algorithm under attack, due to the transferability among many ML algorithms [55]. In a black-box attack, the adversary initiates the assault on an ML classifier without knowledge of the training data. To initiate a black-box attack the agent sends a set of samples to the algorithm and receives the output labels. Then, using the input-output pairs a deep learning (based on a multi-layer neural network) classifier is trained, building a functional copy of the original classifier.

The deep neural network is chosen for its astonishing modeling capacity in pattern recognition. It displays a strong ability to fit the data [22]. Moreover, [82] illustrates that a deep neural network can be used to build a functionally equivalent classifier to a Naive Bayes or an SVM classifier. In addition to that, [67] illustrates how to use a deep neural network to steal a real-life classifier through polling its API (Application Programming Interface) with the use of a free license allowance and a dataset scraped off the internet. The thing of importance in case of black-box attacks is that the adversarial objective is to train a local version of a classifier [15].

On the other hand, the white-box case can be evaluated from a number of angles. The architecture of the model can vary significantly with a myriad of hyper-parameter setups. The knowledge of the specific values can be used to find vulnerabilities which can then be abused by the attacker. A well-informed antagonist can alter the input data to steer the algorithm into spaces where its performance is weak. [55] In a white-box attack, the adversary is assumed to have total knowledge of the inner workings of an ML algorithm, that means the type of ML used, parameters like the number of hidden layers or the number of neurons, as well as the hyper-parameter setups, like the optimizer. The parameters of a deployed model are also assumed to be revealed to him. This knowledge is then used to find the areas where the classifier is open to attack. The knowledge of model weights can be translated into an immensely powerful attack [15].

Model Extraction or model stealing disregards the confidentiality of ML, allowing the adversary to create a 'surrogate model' for the attacked ML algorithm [61].

3.3. Experiments

The aim of this section is to evaluate if it is possible to steal a classifier in the cybersecurity domain by probing an established ML algorithm with a batch of data and training a deep neural network (DNN) on the observed responses. This kind of attack could be a first step to launching more sophisticated adversarial attacks, like poisoning attacks or evasion attacks. Having a local version of a classifier is, therefore, a valuable commodity for a malevolent user. It is important to mention that this work is preliminary research and at this stage, the stolen algorithm is not a live, operational system, but an artificial neural network trained on one of the benchmark cybersecurity datasets. The details of the setup are disclosed in a later section. To make the experiment as real-life as possible under the current circumstances, a set of assumptions was made.

- In a real-life cybersecurity situation, the only observable response would be the restriction of access for an agent displaying a given set of behaviours. This is in a way similar to the oracle attack known from cryptography [10]. Therefore the extracted model will only be able to perform binary classification.
- To make it possible to compare the extracted model with the original algorithm will be a binary classifier as well, only making the distinction between normal and anomalous traffic.
- As the research progresses, attempts will be made to steal a classifier in a completely black-box manner, at this stage however some initial knowledge of the algorithm is assumed. More precisely, the work has been conducted using the NSL-KDD dataset.

3.3.1. Original Classifier

The original classifier was a multi-layer artificial neural network (ANN) trained on the NSL-KDD [73] dataset.

A multi-layer ANN refers to a network with multiple computational layers, known as the hidden layers. The computations in those layers are not perceivable from the users' point of view. The data are fed forward from the input layer throughout the consecutive layers until they reach the output layer.

3.3.2. Model Extraction

We have used a multilayer perceptron as the model extraction architecture, with 4 hidden layers of 512 neurons each and the Rectified Linear Unit activation function. This is motivated by the fact that a deep learning classifier achieved superior results for model extraction of an SVM and a Naive Bayes classifier in [82]. As depicted in [82], the model extraction algorithm procedure can be summarised as follows:

Stealing an algorithm:

1. polling the classifier with input data
2. observing the labels returned by the classifier
3. using input/label data to train a deep learning classifier and optimize its hyperparameters

The pipeline of the process is also depicted in Fig. 2.

3.3.3. Experimental Process

To properly test the feasibility of an algorithm trained on the labels inferred from another algorithm we have prepared the experiment as follows:

Firstly, the dataset was transformed into a binary classification problem. To achieve this the classes contained in the set were converted to 'attack' or 'benign' respectively. The dataset contains 58628 'attack' records and 67342 'benign' records, that constitutes roughly 46.5% attacks and 53,5% benign datapoints.

Secondly, the binary NSL-KDD dataset was split into three parts:

- Set A - used to train the original algorithm
- Set B - used to poll the original classifier and receive classification labels, and then to train the new DNN Classifier
- Set C - used to test the DNN and compare the results with the original classifier (fig.14 and fig.15)

Thirdly, the original algorithm is polled with the features from Set B, the responses are recorded, paired with their respective polls and formed into a new dataset. This new dataset is then utilised to train a DNN.

Finally, we use the remainder of the dataset (Set C) to test the DNN and also feed it to the original classifier to compare the results.

Deep Neural Network architecture

According to [1], a shallow neural network of one layer and sufficient number of neurons has, in theory, the ability to fit to any function. The same source suggests going for depth is also a viable alternative (although not without its own problems). Following in the footsteps of [82], where authors show they are

Tablica 14: Original Classifier's confusion matrix on subset C

	benign	attack
benign	20125	78
attack	76	17512

Tablica 15: Extracted Classifier's confusion matrix on subset C

	benign	attack
benign	20108	95
attack	120	17468

capable of stealing a Naive Bayes and an SVM classifier using a DNN, the authors have used a network of 4 dense layers, 512 neurons each and a Rectified Linear Unit activation function. The architecture was chosen so as to be more complex than the original classifier, taking an educated guess based on the typical number of features in cybersecurity datasets.

Recently it has come to attention that skilfully crafted inputs can affect artificial intelligence algorithms to sway the classification results in the fashion tailored to the adversary needs [16]. This new disturbance in the proliferation of Machine Learning has not yet been extensively researched, and thus the awareness of the challenge is adequately infrequent. At the time of writing this paper a variety of vulnerabilities have been uncovered [16].

With the recent spike of interest in the field of securing ML algorithms, a myriad of different attack and defence methods have been discovered; no truly safe system has been developed however, and no genuinely field-proven solutions exist [46].

The solutions known at this point seem to work for certain kinds of attacks, but do not assure safety against all kinds of adversarial attacks. In certain situations, implementing those solutions could lead to the deterioration of ML performance [16].

There are a couple of known poisoning attacks featured in the literature. In [5] a method utilising the intrinsic properties of Support Vector Machines is introduced. The overarching idea is that an adversary can craft a data point that significantly deteriorates the performance of the classifier. The formulation of that data point can be, as demonstrated by the authors, defined as the solution of an optimisation problem with regard to a performance measure. Thus, gradient ascent is used to identify local maxima of the error surface. The paper introduces a model that analyses label flipping attacks on support vector machines (SVM) in binary classification, which they call adversarial label noise. In their paper, the

authors evaluate two major attack strategies - random label flips and adversarial label flips. Random flips are simply accidental noise, which influences a given percentage of data. The second instance features an adversary seeking the maximisation of classification error on testing data. The testing data has not been tampered with. The authors note that the challenge of finding the worst possible mix of label flips is not a straightforward one. The labels that are flipped the earliest are the ones that carry non-uniform probabilities according to the SVM trained on the clear dataset. The classes chosen for the flips are the ones classified with a high confidence, this should result in a significant impact on SVM accuracy[5].

In [65] the authors investigate a poisoning attack geared towards targeting specific test instances with the ability to fool a labelling authority, which they name 'clean-label' attacks. Their work does not assume knowledge of the training data, but does require the knowledge of the model. It is an optimisation-based attack for both the transfer-learning and end-to-end DNN training cases. The overall procedure of the attacks, called 'Poison Frogs' by the authors, is as follows: the basic version of this attack starts with choosing the target datapoint, then making alterations to that datapoint to make it seem like it belongs to the base class. A poison crafted that way is then inserted into the dataset. The objective is met if the target datapoint is classified as the base class at test time. Arriving at a poisonous datapoint to be inserted into the training set comes as a result of a process called 'feature collision'. It is a process that exploits the nonlinear complexity of the function propagating the input through the second-to-last layer of the neural network to find a datapoint which 'collides' with the target datapoint, but is also close to the base class in the feature space. This allows the poisoned datapoint to bypass the scrutiny of any labelling authority, and also remain in the target class distribution. The optimisation is performed with a forward-backward-splitting iterative procedure.

A targeted backdoor attack is proposed in [18]. The premise of the method is to create a backdoor to an authentication system based on artificial intelligence, allowing the adversary to pass the authentication process by deceiving it. The poisoning datapoints are created specifically to force an algorithm to classify a specific instance as a label of the attacker's choice. The authors propose a method that works with relatively small poison samples and with the adversary possessing no knowledge of the algorithm utilised. This claim is backed up by a demonstration of how inserting just 50 samples gets a 90% success rate.

Intrusion Detection and the ability to detect attacks is a crucial aspect to ensure cybersecurity. However, what if IDS (Intrusion Detection System) is attacked; in other words what defends the defender? In this work, we focus on countering attacks on machine learning-based cyberattack detectors. In principle, we propose the adversarial machine learning detection solution. Indeed, contemporary

machine learning algorithms have not been designed bearing in mind the adversary nature of the environments they are deployed in. Thus, Machine Learning solutions are currently a target of a range of attacks. This paper evaluates the possibility of deteriorating the performance of a well-optimised intrusion detection algorithm at test time by crafting adversarial attacks with the four of the recently proposed methods and then offers a way to detect those attacks. To the best of our knowledge, detecting adversarial attacks on artificial neural network has not yet been widely researched in the context of intrusion detection systems.

3.3.4. Adversarial Machine Learning and Attack Generation

Over the last few years, the research into the curious properties of ML has exploded. The fact that skilfully crafted feature vector can fool even the classifiers that exceed human performance on a benchmark dataset has riveted the attention of the AI scientific community. With the awareness of the issue rising, a range of soft spots has been found [15]. Adversarial examples are samples that for all intents and purposes look almost identical to correctly classifiable data; however with a small, intentional, worst-case perturbation that can cause a range of ML algorithms, most notably artificial neural networks, to fail [72].

Fast Gradient Sign Method

The authors of [25] found a rapid approach to dependably produce adversarial examples that lead an array of ML methods to misclassify. The method was initially demonstrated on ImageNet, MNIST [42] and CIFAR-10 [37] datasets. It relies on finding a small adversarial noise vector that when summed up corresponds with the sign of the elements of the gradient of the cost function for the evaluated sample. The Fast Gradient Sign Method can be defined as the linearization of the cost function around the current value of Θ, obtaining an optimal max-norm constrained perturbation of

$$\eta = \epsilon sign(\nabla_x J(\Theta, x, y))$$

Where Θ stands for the parameters of the model, x for the inputs to the model, y for the targets of the corresponding x and $J(\Theta, x, y)$ is the cost utilised to train the ANN [25]. The method is referred to as Fast Gradient Sign Method (FGSM), Fast Gradient Method (FGM) or simply Fast Method in literature.

Basic Iterative Method

The authors of [40] offer an extension to the FGM method, applying it multiple times using a small step size, clipping the values after every transitional pace, also using $\alpha = 1$, which translates to changing the value of each element (i.e. pixel) by 1. In their work the authors have chosen the number of iterations heuristically

to be enough to reach the border of the ϵ max-norm ball. The formula used is as follows:

$$X_0^{adv} = X_1,\ X_{N+1}^{adv} = Clip_{X,\epsilon}\{X_N^{adv} + \alpha sign(\nabla_x J(X_N^{adv}, y_{true}))\}$$

Carlini and Wagner Attack

[14] offer the solution to the adversarial example creation by formulating the optimisation problem in a way that can be dealt with by current algorithms. The optimisation problem is formally defined as

$$\begin{aligned} minimise \quad & D(x, x+\delta) \\ such\,that \quad & C(x+\delta) = t \\ & x + \delta \in [0,1]^n \end{aligned}$$

where x does not change, so one aims to find δ that minimises $D(x, x+\delta)$. In other words - finding δ that will change the classification. D is a distance metric; in their paper the authors evaluate three of them - L_0, L_2 and L_∞, however for the use in this paper L_2 was selected.

To make the formula solvable the authors redefine an objective function f so that $C(x+\delta) = t$ when $f(x+\delta) \leq 0$ and offer a range of options for the f formula. In this work the attack defined in [14] as the L_2 attack was used.

Projected Gradient Descent

In [47] Projected Gradient Descent is put forward as the strongest attack and the universal 'first order adversary', as it is the definitive method for constrained large-scale optimization. Essentially, the abovementioned FGM is a one-step method for generating adversarial examples, in theory a more dangerous option would be the multi-step procedure, which the authors call 'essentially projected gradient descent', formulated as follows:

$$X^{t+1} = \Pi_{x+S}(x^t + \alpha sgn(\nabla_x L(\Theta, x, y)))$$

Summary of Adversarial Attack Generation

Following the summary found in [47], the attack model can be definitively formulated as a two-level optimisation problem, expressed by the following:

$$\min_\theta \rho(\Theta),\ where\, \rho(\Theta) = E_{(x,y)\sim D}[\max_{\delta \in S} L(\Theta, x+\delta, y)]$$

where S is the set of allowed perturbations, D is the distribution, L is the loss and $E_{(x,y)\sim D}$ is the perturbed input before the sample is fed to the loss. This formulation allows to consider the $inner\ maximisation$ and the $outer$

$minimisation$ problems. In essence, the four abovementioned attacks are four different approaches to solving this formula.

3.3.5. Countering Adversarial Attacks

A number of possible defences against the effects of adversarial examples have been put forward. One of the defences to be proposed is adversarial retraining, either by trying to correctly classify the adversarial example itself [72, 25, 44, 27] or by creating a separate class for adversarial examples.

This method has its merits, but is not effective on unforeseen attacks, and causes a deterioration of the model in many applications.

There are some researchers who propose training a second classifier to detect Adversarial Examples [24]. The authors claim robustness to FGM and Jacobian Saliency Map Attacks [56]. The approach, however, learns to distinguish adversarial examples from the non-adversarial ones using the same distribution and thus can be evaded by formulating the attack to find adversarial examples that fool both classifiers at the same time, as demonstrated by [13].

The general consensus among the researches with regard to the defensive measures is that no fully safe system has been put forward and no truly field-proven solutions exist [45]. The methods developed to this point apply to certain kinds of attacks, but do not provide defence against all possible adversarial attacks. Some of those solutions lead to the deterioration of ML performance [15].

3.4. Proposed Method for Evasion Attack Detection in Neural Networks

dopiska krótkiej zajawki: In this section, the overall approach for Evasion Attack Detection in Neural Networks will be presented. Firstly, the utilised dataset is disclosed, which is followed by the applied dataset preprocessing and the IDS training pipeline. Then the attacks on the IDS are performed and tested. The neural activations of those attacks, as well as the activations for clear samples are gathered; finally the attack detector is trained and tested.

In this work, the CICIDS2017 dataset was used [66]. The dataset was first cut into four parts in a stratified fashion to ensure full coverage of all kinds of attacks included in the dataset in all its sub-parts. This procedure results in the following setup:

- Dataset A - used to train the IDS classifier
- Dataset B - used to test the IDS classifier and to craft the adversarial attacks and test them on the original IDS ANN, then to acquire the activations of neural nodes in the IDS network of benign, attack and adversarial samples to train the Adversarial Detector

Tablica 16: 'IDS ANN' trained on Dataset A and tested on Dataset B

	precision	recall	f1-score	support
ATTACK	0.96	0.97	0.97	139675
BENIGN	0.99	0.99	0.99	405383
micro avg	0.98	0.98	0.98	545058
macro avg	0.97	0.98	0.98	545058
weighted avg	0.98	0.98	0.98	545058
samples avg	0.98	0.98	0.98	545058

– Dataset C and D - used to craft test adversarial samples and acquire the activations for the neural nodes of benign, attack and adversarial samples

All of the sub-parts were then turned into a binary classification task, leaving all the benign samples as 'BENIGN', but changing all the names of possible attacks to simply 'ATTACK'.

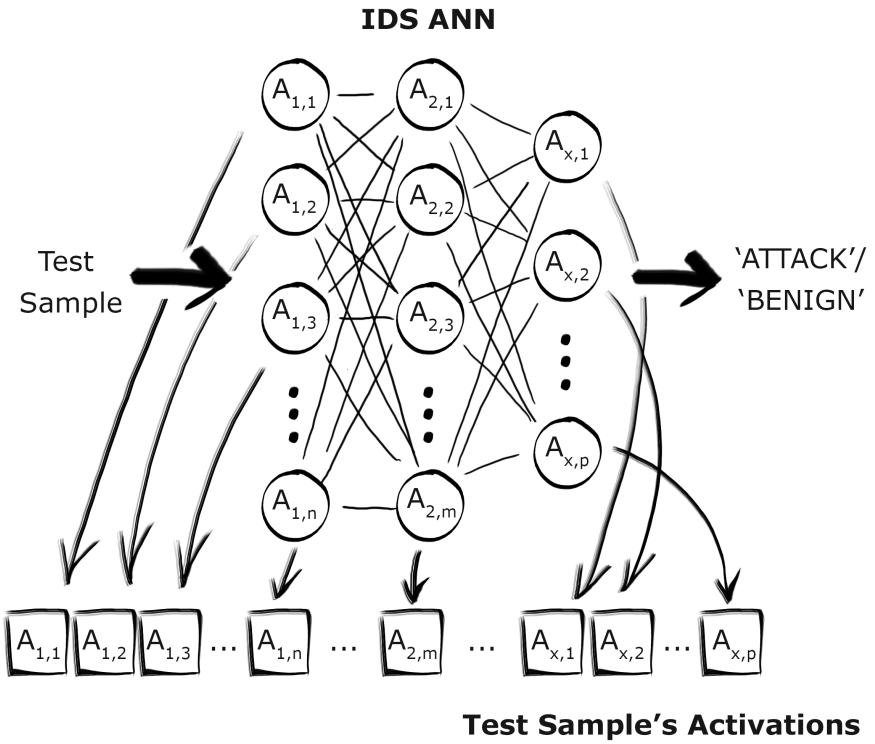
The IDS setup was as follows: An Artificial Neural Network of 3 hidden layers was compiled, with 40 neurons on the first hidden layer, 40 on the second and 20 on the third layer. The Rectified Linear Unit activation function was utilised and the optimiser selected was ADAM. With batch size of 100 and 10 epochs the network achieved an accuracy of 0.9827 when trained with Dataset A and tested on Dataset B. The precision, recall and f1-score are showcased in Tab. 16 As seen in the figure, the binarized dataset is fed to the architecture described above, and the training procedure results in building a model capable of binary classification.

3.4.1. Evasion Adversarial Attacks

After testing the trained IDS (the optimisation procedure of an ANN-based IDS can be found in [59]) four different adversarial attacks were crafted based on the ATTACK class of Dataset B. The algorithms used for the creation of evasion attacks were:

– Carlini and Wagner attack (CW) [14]
– Fast Gradient Sign Method (FGM) [25]
– Basic Iterative Method (BIM) [40]
– Projected Gradient Descent (PGD) [47]

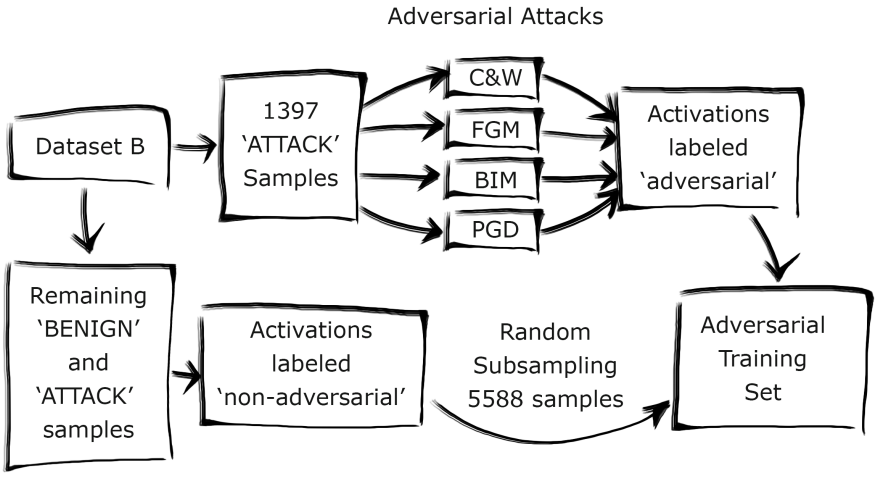
1397 samples of the 'ATTACK' class were randomly extracted from Dataset B and turned into adversarial samples with the use of those four algorithms. The IDS ANN classified those as 1353 ATTACKS and 44 BENIGNS. Using adversarial attacks we were able to force the IDS ANN to classify 1296 'ATTACK' as 'BENIGN' samples for BIM and PGD, 1324 for FGM and 59 for CW. The zeros



Rysunek 4: The acquisition of IDS ANN activations for a given test sample

and the results of the zeros compared to original results The abovementioned procedures introduce adversarial noise to the samples. This noise resulted in negative values in some of the features. Those negative values were supplanted by zeros with some loss of effectiveness of those attacks - BurntOrange (55 more attacks were classified as benign samples with the original BIM and PGD methods, 295 more for CW and interestingly, 21 fewer for FGM.

The samples from Dataset B not used in crafting Adversarial Attacks were annotated as 'nonadversarial', the Adversarial Attacks were labelled 'adversarial'. With 5588 adversarial attack samples, a matching number of nonadversarial records was randomly picked from unused samples of Dataset B to form the base for a balanced 'Adversarial Training Dataset' for the adversarial attack detector. The procedure is depicted in Fig. 5. Dataset D was subjected, except for the balancing, to the exact same crafting/annotation procedure to form the base for the testing dataset for the detector.



Rysunek 5: Forming the Adversarial Training Dataset from Dataset B

3.4.2. Detection Method

The training and testing activation datasets were fed to the IDS ANN and the activations for all 102 neurons (including the softmax layer), as shown in Fig.4, were recorded and annotated as adversarial or nonadversarial respectively.

The recorded activations were used to train the detector artificial neural network. The architecture of the detector is as follows: 3 hidden layers with the ReLU activation function, 51, 51 and 25 neurons respectively and the ADAM optimiser. Using batch size of 100 and just 10 epochs, the detector achieved an accuracy of 0.8506 on the testing set. The detailed results are assembled in Tab. 17.

3.5. The Evaluation of the Adversarial Attack Detector

The detector achieves high accuracy and the recall for the adversarial class signifies that it can recognise the attacks with great promise. The precision however is the evidence of a high number of false-positives.

Since in the process of creating the ANN-based adversarial attack detector a dataset of neural activations of the IDS architecture was created, the authors proceeded to test the approach using other well-established classifiers. In Tab. 18 the results of detection with Random Forest (RF) are presented. As immediately apparent, with this kind of data RF achieves results superior to ANN, with higher

Tablica 17: Results of ANN-based Adversarial Attack Detector over the test set activations

	precision	recall	f1-score	support
adversarial	0.06	0.91	0.11	5588
non-adversarial	1.00	0.85	0.92	543661
micro avg	0.85	0.85	0.85	549249
macro avg	0.53	0.88	0.51	549249
weighted avg	0.99	0.85	0.91	549249
samples avg	0.85	0.85	0.85	549249

Tablica 18: Results of Random-Forest-based Adversarial Attack Detector over the test set activations

	precision	recall	f1-score	support
adversarial	0.11	0.99	0.20	5588
nonadv	1.00	0.91	0.95	543661
micro avg	0.92	0.91	0.92	549249
macro avg	0.56	0.95	0.58	549249
weighted avg	0.99	0.91	0.95	549249
samples avg	0.91	0.91	0.91	549249

recall and better precision. Notably, the accuracy of this approach exceeds 91% (91.24).

Following the success of the RF-based classifier, another ensemble method was tested. The ADABoost algorithm did not surpass the results of the Random Forest, getting up to only 87.66% accuracy. This, however, is still a better result than the ANN approach. The details can be found in Tab. 19

The tests were then followed by building a model relying on the Support Vector Machine algorithm. As can be noticed from investigating the Tab. 20,

Tablica 19: Results of ADABoost-based Adversarial Attack Detector over the test set activations

	precision	recall	f1-score	support
adversarial	0.07	0.90	0.13	5588
nonadv	1.00	0.88	0.93	543661
macro avg	0.53	0.89	0.53	549249
weighted avg	0.99	0.88	0.93	549249

Tablica 20: Results of an SVM-based Adversarial Attack Detector over the test set activations

	precision	recall	f1-score	support
adversarial	0.11	0.79	0.19	5588
nonadv	1.00	0.93	0.97	543661
macro avg	0.55	0.86	0.58	549249
weighted avg	0.99	0.93	0.96	549249

Tablica 21: The results of the evasion attack detector based on the nearest neighbour algorithm

	precision	recall	f1-score	support
adversarial	0.11	0.99	0.20	5588
nonadv	1.00	0.91	0.95	543661
macro avg	0.56	0.95	0.58	549249
weighted avg	0.99	0.91	0.95	549249

the SVM was not as successful in picking up on the adversarial attacks based on the activations as the other algorithms, with the recall of only 0.79.

Finally, the same activation dataset was utilised to train a nearest neighbour classifier. The procedure achieved results on par with the Random Forest method.

4. Conclusion

This thesis comprises of two parts, both offering innovative solutions for network intrusion detection based on machine learning. The first one is a modification of machine learning methods to be applied specifically in IDS, the other one geared toward detecting attacks on machine learning itself.

The first part of this work showcases the influence dataset balancing methods and hyperparameter optimisation has over the results ML achieves in IDS applications. A Gated Recurrent Unit Neural Network was tested in IDS, and a new method was proposed and experimentally tested on a new dataset containing relevant data. The accuracy of the proposed methods exceeded 99%.

The second part of this work focuses on adversarial attacks, and introduces a novel method of detecting evasion attacks. The approach is capable of detecting 99% of evasion attacks against IDS. However, the high false positive rate indicates that further improvements are possible. In this part the model extraction procedure in cybersecurity was also conducted.

The results of conducted experiments prove that the thesis formulated at the beginning of this work is confirmed.

References

[1] Charu C. Aggarwal. *Neural Networks and Deep Learning A Textbook*. Springer, Cham, 2018.

[2] Tomasz Andrysiak, Łukasz Saganowski, Michał Choraś, and Rafał Kozik. Network traffic prediction and anomaly detection based on ARFIMA model. In *Advances in Intelligent Systems and Computing*, pages 545–554. Springer International Publishing, 2014.

[3] Giuseppe Ateniese, Giovanni Felici, Luigi V. Mancini, Angelo Spognardi, Antonio Villani, and Domenico Vitali. Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers. *CoRR*, abs/1306.4447, 2013.

[4] Agnieszka Bielec. ąnalysis of a polish bankbot".

[5] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. *arXiv preprint arXiv:1206.6389*, 2012.

[6] Barbara Bobowska, Michał Choraś, and Michał Woźniak. Advanced analysis of data streams for critical infrastructures protection and cybersecurity. *J. UCS*, 24(5):622–633, 2018.

[7] Samarjeet Borah, Ranjit Panigrahi, and Anindita Chakraborty. An enhanced intrusion detection system based on clustering. In *Advances in Intelligent Systems and Computing*, pages 37–45. Springer Singapore, dec 2017.

[8] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001.

[9] Charu C. Aggarwal. *Neural Networks and Deep Learning: A Textbook*. 01 2018.

[10] Christian Cachin and Jan Camenisch. Advances in cryptology - eurocrypt 2004, international conference on the theory and applications of cryptographic techniques, interlaken, switzerland, may 2-6, 2004, proceedings. 3027, 01 2004.

[11] G. Canfora, A. Di Sorbo, F. Mercaldo, and C. A. Visaggio. Obfuscation techniques against signature-based detection: A case study. In *2015 Mobile Systems Technologies Workshop (MST)*, pages 21–26, May 2015.

[12] Alvaro A. Cardenas, Saurabh Amin, and Shankar Sastry. Secure control: Towards survivable cyber-physical systems. In *2008 The 28th International Conference on Distributed Computing Systems Workshops*. IEEE, jun 2008.

[13] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected. *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security - AISec '17*, 2017.

[14] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57. IEEE, 2017.

[15] Anirban Chakraborty, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. Adversarial attacks and defences: A survey. *CoRR*, abs/1810.00069, 2018.

[16] Anirban Chakraborty, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. Adversarial attacks and defences: A survey. *arXiv preprint arXiv:1810.00069*, 2018.

[17] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: Synthetic minority over-sampling technique. *J. Artif. Int. Res.*, 16(1):321–357, June 2002.

[18] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017.

[19] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[20] M. Choraś and R. Kozik. Machine learning techniques applied to detect cyber attacks on web applications. *Logic Journal of the IGPL*, 23(1):45–56, Feb 2015.

[21] Michał Choraś, Rafał Kozik, Damian Puchalski, and Witold Hołubowicz. Correlation approach for SQL injection attacks detection. In *Advances in Intelligent Systems and Computing*, pages 177–185. Springer Berlin Heidelberg, 2013.

[22] Ivan Nunes da Silva · Danilo Hernane Spatti Rogerio Andrade Flauzino Luisa Helena Bartocci Liboni Silas Franco dos Reis Alves. *Artificial Neural Networks A Practical Course*. 2017.

[23] W. Gong, W. Fu, and L. Cai. A neural network based intrusion detection data fusion model. In *2010 Third International Joint Conference on Computational Science and Optimization*, volume 2, pages 410–414, May 2010.

[24] Zhitao Gong, Wenlu Wang, and Wei-Shinn Ku. Adversarial and clean data are not twins, 2017.

[25] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2014.

[26] Palash Goyal, Sumit Pandey, and Karan Jain. *Unfolding Recurrent Neural Networks*, pages 119–168. Apress, Berkeley, CA, 2018.

[27] Kathrin Grosse, Praveen Manoharan, Nicolas Papernot, Michael Backes, and Patrick McDaniel. On the (statistical) detection of adversarial examples, 2017.

[28] F. Haddadi, S. Khanchi, M. Shetabi, and V. Derhami. Intrusion detection and attack classification using feed-forward neural network. In *2010 Second International Conference on Computer and Network Technology*, pages 262–266, April 2010.

[29] Y. Hao, Y. Sheng, and J. Wang. Variant gated recurrent units with encoders to preprocess packets for payload-aware intrusion detection. *IEEE Access*, 7:49985–49998, 2019.

[30] Raymond C. Borges Hink, Justin M. Beaver, Mark A. Buckner, Tommy Morris, Uttam Adhikari, and Shengyi Pan. Machine learning for power system disturbance and cyber-attack discrimination. In *2014 7th International Symposium on Resilient Control Systems (ISRCS)*. IEEE, aug 2014.

[31] Nwokedi Idika and Aditya Mathur. A survey of malware detection techniques. *Purdue University*, 03 2007.
[32] Witten D. Hastie T. & Tibshirani R. James, G. An introduction to statistical learning. In *Cluster Comput (2018).*, 2013.
[33] Leo Kelion. ębay redirect attack puts buyers' credentials at risk".
[34] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, pages 1137–1145, 1995.
[35] Rafał Kozik and Michał Choraś. Solution to data imbalance problem in application layer anomaly detection systems. pages 441–450, 04 2016.
[36] Choraś M. Kozik R. Solution to data imbalance problem in application. *Martinez-Alvarez F., Troncoso A., Quintian H., Corchado E. (Eds.): Hybrid Artificial Intelligent Systems,*, LNAI vol. 9648:441–450, 2016.
[37] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 55, 2014.
[38] R. Kumar Singh Gautam and E. A. Doegar. An ensemble approach for intrusion detection system using machine learning algorithms. In *2018 8th International Conference on Cloud Computing, Data Science Engineering (Confluence)*, pages 14–15, Jan 2018.
[39] Kunal and M. Dua. Machine learning approach to ids: A comprehensive review. In *2019 3rd International conference on Electronics, Communication and Aerospace Technology (ICECA)*, pages 117–121, June 2019.
[40] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.
[41] T. Le, H. Kang, and H. Kim. The impact of pca-scale improving gru performance for intrusion detection. In *2019 International Conference on Platform Technology and Service (PlatCon)*, pages 1–6, Jan 2019.
[42] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.
[43] Dave Lee. "myfitnesspal breach affects millions of under armour users".
[44] Bo Li, Yevgeniy Vorobeychik, and Xinyun Chen. A general retraining framework for scalable adversarial classification, 2016.
[45] X. Liao, L. Ding, and Y. Wang. Secure machine learning, a brief overview. In *2011 Fifth International Conference on Secure Software Integration and Reliability Improvement - Companion*, pages 26–29, June 2011.
[46] Xiaofeng Liao, Liping Ding, and Yongji Wang. Secure machine learning, a brief overview. In *2011 Fifth International Conference on Secure Software Integration and Reliability Improvement-Companion*, pages 26–29. IEEE, 2011.
[47] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks, 2017.
[48] Oded Maimon and Lior Rokach. *Data Mining and Knowledge Discovery Handbook, 2nd ed.* 01 2010.
[49] Gary McGraw and Greg Morrisett. Attacking malicious code: A report to the infosec research council. *IEEE Softw.*, 17(5):33–41, September 2000.
[50] I. Mukhopadhyay, M. Chakraborty, S. Chakrabarti, and T. Chatterjee. Back propagation neural network approach to intrusion detection system. In *2011 International Conference on Recent Trends in Information Systems*, pages 303–308, Dec 2011.

[51] Paul Mutton. "hackers still exploiting ebay's stored xss vulnerabilities in 2017".

[52] S. Naseer, Y. Saleem, S. Khalid, M. K. Bashir, J. Han, M. M. Iqbal, and K. Han. Enhanced network anomaly detection based on deep neural networks. *IEEE Access*, 6:48231–48246, 2018.

[53] K. D. T. Nguyen, T. M. Tuan, S. H. Le, A. P. Viet, M. Ogawa, and N. L. Minh. Comparison of three deep learning-based approaches for iot malware detection. In *2018 10th International Conference on Knowledge and Systems Engineering (KSE)*, pages 382–388, Nov 2018.

[54] K. Ozkan, S. Isik, and Y. Kartal. Evaluation of convolutional neural network features for malware detection. In *2018 6th International Symposium on Digital Forensic and Security (ISDFS)*, pages 1–5, March 2018.

[55] N. Papernot, P. McDaniel, A. Sinha, and M. P. Wellman. Sok: Security and privacy in machine learning. In *2018 IEEE European Symposium on Security and Privacy (EuroSP)*, pages 399–414, April 2018.

[56] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. *2016 IEEE European Symposium on Security and Privacy (EuroSP)*, Mar 2016.

[57] Simone Bassis Anna Esposito Francesco Carlo Morabito Eros Pasero. *Advances in Neural Networks*. 2016.

[58] T. M. Pattewar and H. A. Sonawane. Neural network based intrusion detection using bayesian with pca and kpca feature extraction. In *2015 IEEE International Conference on Computer Graphics, Vision and Information Security (CGVIS)*, pages 83–88, Nov 2015.

[59] Marek Pawlicki, Rafał Kozik, and Michał Choraś. Artificial neural network hyperparameter optimisation for network intrusion detection. In *Intelligent Computing Theories and Application - 15th International Conference, ICIC 2019, Nanchang, China, August 3-6, 2019, Proceedings, Part I*, pages 749–760, 2019.

[60] J. D. J. S. Pérez, M. S. Rosales, and N. Cruz-Cortés. Universal steganography detector based on an artificial immune system for jpeg images. In *2016 IEEE Trustcom/BigDataSE/ISPA*, pages 1896–1903, Aug 2016.

[61] E. Quiring, D. Arp, and K. Rieck. Forgotten siblings: Unifying attacks on machine learning and digital watermarking. In *2018 IEEE European Symposium on Security and Privacy (EuroSP)*, pages 488–502, April 2018.

[62] K. Rahul 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)*, pages 1–6, July 2018.

[63] Łukasz Saganowski, Marcin Goncerzewicz, and Tomasz Andrysiak. Anomaly detection preprocessor for SNORT IDS system. In *Advances in Intelligent Systems and Computing*, pages 225–232. Springer Berlin Heidelberg, 2013.

[64] Y. Sani, A. Mohamedou, K. Ali, A. Farjamfar, M. Azman, and S. Shamsuddin. An overview of neural networks use in anomaly intrusion detection systems. In *2009 IEEE Student Conference on Research and Development (SCOReD)*, pages 89–92, Nov 2009.

[65] Ali Shafahi, W Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison frogs! targeted clean-label poisoning

attacks on neural networks. In *Advances in Neural Information Processing Systems*, pages 6103–6113, 2018.

[66] Iman Sharafaldin., Arash Habibi Lashkari., and Ali A. Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *Proceedings of the 4th International Conference on Information Systems Security and Privacy - Volume 1: ICISSP,*, pages 108–116. INSTICC, SciTePress, 2018.

[67] Yi Shi, Yalin E. Sagduyu, Kemal Davaslioglu, and Jason H. Li. Generative adversarial networks for black-box API attacks with limited training data. *CoRR*, abs/1901.09113, 2019.

[68] Sandro Skansi. *Introduction to Deep Learning: from logical calculus to artificial intelligence*. Springer, 2018.

[69] Sandro Skansi. *Recurrent Neural Networks*, pages 135–152. 01 2018.

[70] H. A. Sonawane and T. M. Pattewar. A comparative performance evaluation of intrusion detection based on neural network and pca. In *2015 International Conference on Communications and Signal Processing (ICCSP)*, pages 0841–0845, April 2015.

[71] B. Subba, S. Biswas, and S. Karmakar. A neural network based system for intrusion detection and attack classification. In *2016 Twenty Second National Conference on Communication (NCC)*, pages 1–6, March 2016.

[72] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[73] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A Ghorbani. A detailed analysis of the kdd cup 99 data set. In *2009 IEEE symposium on computational intelligence for security and defense applications*, pages 1–6. IEEE, 2009.

[74] Fredrik Valeur. *Real-Time Intrusion Detection Alert Correlation*. PhD thesis, UNIVERSITY OF CALIFORNIA, 2006.

[75] N. T. T. Van and T. N. Thinh. Accelerating anomaly-based ids using neural network on gpu. In *2015 International Conference on Advanced Computing and Applications (ACOMP)*, pages 67–74, Nov 2015.

[76] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman. Deep learning approach for intelligent intrusion detection system. *IEEE Access*, 7:41525–41550, 2019.

[77] Eric Ke Wang, Yunming Ye, Xiaofei Xu, S. M. Yiu, L. C. K. Hui, and K. P. Chow. Security issues and challenges for cyber physical system. In *2010 IEEE/ACM Int Conference on Green Computing and Communications & Int Conference on Cyber, Physical and Social Computing*. IEEE, dec 2010.

[78] W. Wang, Y. Sheng, J. Wang, X. Zeng, X. Ye, Y. Huang, and M. Zhu. Hast-ids: Learning hierarchical spatial-temporal features using deep neural networks to improve intrusion detection. *IEEE Access*, 6:1792–1806, 2018.

[79] C. Xu, J. Shen, X. Du, and F. Zhang. An intrusion detection system using a deep neural network with gated recurrent units. *IEEE Access*, 6:48697–48707, 2018.

[80] K. Xu, Y. Li, R. H. Deng, and K. Chen. Deeprefiner: Multi-layer android malware detection system applying deep neural networks. In *2018 IEEE European Symposium on Security and Privacy (EuroSP)*, pages 473–487, April 2018.

[81] M. Yeo, Y. Koo, Y. Yoon, T. Hwang, J. Ryu, J. Song, and C. Park. Flow-based malware detection using convolutional neural network. In *2018 International Conference on Information Networking (ICOIN)*, pages 910–913, Jan 2018.

[82] Yi Shi, Y. Sagduyu, and A. Grushin. How to steal a machine learning classifier with deep learning. In *2017 IEEE International Symposium on Technologies for Homeland Security (HST)*, pages 1–5, April 2017.

[83] Ying Wang, Yongjun Shen, and Guidong Zhang. Research on intrusion detection model using ensemble learning methods. In *2016 7th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, pages 422–425, Aug 2016.

[84] X. Yuan. Phd forum: Deep learning-based real-time malware detection with multi-stage analysis. In *2017 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 1–2, May 2017.

List of Figures

List of Tables