Flow-Based Intrusion Detection for SCADA networks using Supervised Learning

Gabriel Vasquez[1], **Rodrigo S. Miani** [2], **Bruno B. Zarpelão**[1]

[1]Computer Science Department - State University of Londrina (UEL)
Londrina, PR - Brazil

[2]School of Computer Science - Federal University of Uberlândia (UFU)
Uberlândia, MG - Brazil

gabriel@vasquez.com.br, miani@ufu.br, brunozarpelao@uel.br

Abstract. *Recent attacks on industrial networks have brought the question of their protection, given the importance of the equipment that they control. In this paper, we address the application of Machine Learning (ML) algorithms to build an Intrusion Detection System (IDS) for these networks. As network traffic usually has much less malicious packets than normal ones, intrusion detection problems have class imbalance as a key characteristic, which can be a challenge for ML algorithms. Therefore, we study the performance of nine different ML algorithms in classifying IP flows of an industrial network, analyzing the impact of class imbalance in the results. The algorithms were evaluated taking as main metrics the F1-Score and Averaged Accuracy. Our experiments showed that the three algorithms based on decision trees were superior to the others. Particularly, the Decision Jungle algorithm outperformed all the others.*

1. Introduction

Industrial control systems (ICS) are very common in large industries, as they monitor and control many devices through constant polling. This is accomplished using SCADA (Supervisory Control and Data Acquisition) systems, which usually communicate with the various devices using Modbus Protocol. Recent attacks on these networks have brought much attention to the industry about how to protect those networks [Loukas 2015] [Piggin 2015].

An analysis of incidents in critical infrastructures and other industrial networks in the period of 1982-2012 by Miller and Rowe [2012] shows that the number of attacks has a tendency to rise over the years. Moreover, the main motivation behind those incidents is to disrupt the services provided, hitting both industry and government infrastructures. This shows that the risk of an attack on such networks is increasing continuously.

Piggin [2015] conducted a survey involving 599 companies in which he found that 52% were unaware that industrial networks had vulnerabilities. According to him, we can consider that security in industrial networks is still in its early childhood, meaning that not only their administrators are unconscious of the susceptibility of an attack, but also about the risks that it represents. An example of a serious attack that hit industrial networks was Stuxnet, a malware specifically developed to disrupt this kind of network. Its main target was a Siemens controller used in Iran's nuclear centrifuges, and the main goal was to destroy the uranium-enriching centrifuges [Ntalampiras 2015], [Yusheng et al. 2017].

As ICS are gradually being connected to the Internet, the risks faced by these networks tend to increase even more. Hence, we note that there is a great need for an

Intrusion Detection System (IDS) that can correctly detect attacks against these networks with the lowest number of false positives possible. An anomaly-based IDS using Machine Learning (ML) could provide satisfactory results to the network administrator as regular ICS traffic is related to a limited number of requests and responses, making it clearly different from malicious traffic.

In this paper, we propose that the protection of ICS could be partially accomplished by implementing an Anomaly-based IDS using ML algorithms. This IDS will be responsible for collecting and analyzing the packets that travel through the SCADA network. Therefore, we will gather and analyze Modbus/TCP traffic, aggregate it into IP flows and use this data as an input to several ML algorithms to find the ones that have the best results. We also will study the impacts of class imbalance, which is a common issue in anomaly detection problems. The evaluation will be performed using a dataset provided by Lemay and Fernandez [2016], which is publicly available.

The rest of this paper is organized as follows: Section 2 presents the related work. Section 3 explains the proposed approach. Section 4 presents the dataset, the test environment, the results and discusses the experiments. Lastly, Section 5 presents the conclusions.

2. Related Work

In recent years, many methods have been proposed to aid intrusion detection in many areas, some of them specifically on industrial networks. The intrusion detection usually is divided into tree types: signature-based IDS, in which the detection of the malicious traffic depends on previously created signatures for the malicious traffic; specification-based IDS, in which the detection occurs from differences between the designed normal states of the network and the operational states; and, anomaly-based IDS, in which the detection of the malicious traffic occurs based on the deviations from the normal learned behavior of the network.

In the work of Linda et al. [2009], the authors use two Neural Network algorithms, namely the Error Back-propagation and the Levenberg-Marquardt, to analyze a SCADA network traffic. In their approach, the 8 most relevant features are extracted from the dataset according to a defined window size and merged with artificially generated intrusion traffic to compose the training dataset.

In the work of Junejo and Goh [2016] , the authors developed an anomaly-based IDS to be used on the SWaT water purification testbed. In the work, the authors used nine machine learning algorithms to classify the malicious traffic. Although the authors reached low false positive rates, high precision and recall, the work did not focus on the impacts of the class imbalance of the dataset generated or the benefits of grouping data into flows.

In the work of Yang et al. [2013], it is presented a hybrid approach between signature-based IDS and specification-based IDS for IEC 60870-5-104 networks. The first part of the proposed IDS detects malicious attacks using a set of Snort rules developed by the authors. The second part of the IDS increases the detection capability of the solution by detecting unknown attacks using the specifications of the expected behavior of the entire system. The work of the authors did not present the source of the used dataset or a comparison table to demonstrate that this approach provides relevant results. Also, the definition of the specifications in the second part would require an expert to define the normal states.

Schuster et al. [2015] use One-Class Support Vector Machine (OCSVM) to develop an anomaly based IDS. In their approach, they group each 3 packets from the same origin in a vector. Then the authors employ SVM to discover the normal data distribution on a production ICS dataset in a Profibus network. The authors reach high precision, recall and F-score using different kernels for the SVM.

In the work of Yusheng et al. [2017], the authors developed an IDS for ICS using the ModBus/TCP Protocol. Their approach consists of two modules: rule extraction and deep inspection. The first part dissects the ModBus TCP packet into three subparts, namely network layer, transport layer and application layer. Then, the rule extraction module creates normal and abnormal rules based on the correlation among those three subparts. The deep inspection module is a resident module responsible for continually correlating the classifications to determine whether the normal or abnormal rule current applied is a false positive. It is not clear how the rules would be updated over time to detect new attacks or if the rule extraction module could be susceptible to manipulation on the sliding window process due to slow-rate attacks.

In this work, we choose a different approach than Linda et al. [2009] and Schuster et al. [2015], grouping data into flows, which allows an overview of the stream that is passing through the network, reducing the volume of data to be analyzed. We also choose to use a publicly available dataset differently from Yusheng et al. [2017] and Yang et al. [2013]. This allows the reproducibility of the attained results by future works in the area. The imbalance between normal traffic and anomalous is addressed by this work as well, analyzing the results with more suitable metrics than those used by Junejo and Goh [2016]. At last, the usage of machine learning algorithms may be more efficient in detecting unknown attacks since it does not require the development of signatures for new each attack or the definition of the specifications of the workload as in Yang et al. [2013].

3. Proposed Approach

In this section, we present our approach to detect intrusions in Modbus/TCP SCADA networks using a supervised classifier. Our approach is based on the hypothesis that attacks against a Modbus/TCP SCADA network will cause a significant deviation from normal traffic patterns. Therefore, using machine learning algorithms, we analyze those patterns to detect those variations and notify the network administrator about ongoing attacks.

The proposed system is divided into five parts: the Network Packets Gathering, the Flow Generation, the Feature Creation, the Training, and the Supervised Classification. An overview of the system is presented in Figure 1.

Figure 1. The proposed approach.

The first step of our approach is the Network Packets Gathering. It is responsible for capturing the Modbus/TCP packets. Modbus/TCP [Swales et al. 1999] is the connection-oriented version of traditional Modbus. Although there are other SCADA

protocols like DNP3, IEC-61850 and Profibus, our work is focused specifically on the Modbus/TCP traffic.

The following data from each packet is extracted to compose the initial dataset: a number to identify the packet, the timestamp, the source IP address, the source port, the destination IP address, the destination port, the communication protocol, the length of the packet, and the info field, which contains the information that allows the poll and control of the Modbus devices.

In the second step of our approach, referred to as Flow Generation, the packet data extracted in the first step is used to group the packets into IP flows according to the IP Flow Information Export (IPFIX) Protocol [Claise et al. 2013]. IPFIX protocol specifies that a IP flow is composed of packets with the same source and destinations addresses, source and destination ports, and transport protocol. Besides, the IPFIX protocol suggests a timeout window of 1800 seconds to consider a flow as inactive, which we adopted. The main advantage of the IP flows is that they provide an overview of the network behavior, showing the main statistics of the interactions among the network nodes, which can reveal deviations from the normal behavior caused by attacks. In this work, we consider an IP flow as malicious if any packet of this flow is related to a malicious activity.

The next step of our approach is the Feature Creation. In this step, features are extracted from the IP flows generated in the previous step, creating data instances that will be the input of the supervised classification algorithm. One of the main characteristics of the traffic in SCADA networks is that it is very well established, showing recognizable pattern in the communication among devices. That means that the polling transactions between masters and slaves follow a very foreseeable schedule, and the packets that travel in these networks usually have similar sizes. Based on these particularities, we believe that IP flow features, namely average packet size, source and destination port numbers, and packets per second may indicate that an IP flow is malicious. All the features and their descriptions can be seen in Table 1.

In the Training step, a set of data instances labeled as malicious or legitimate is provided to the machine learning algorithm so it can build a model to classify the future data instances automatically.

In the last step, the Supervised Classification, the machine learning algorithm trained in the previous step is employed to classify the instances produced by the Feature Creation step as malicious or legitimate. Since we are dealing with traffic patterns, we believe that machine learning algorithms will deal with this type of traffic better than signature-based detection systems, specially when dealing with unknown attacks.

Table 1. Flow based dataset features.

Feature	Description
source_port	Source Port, originally created on Flow Generation step, represents the source transport port used on the traffic between two devices.
dest_port	Destination Port, originally created on Flow Generation step, represents the destination transport used on the traffic between two devices.
prot	Transport Protocol, originally created on Flow Generation step, represents the transport protocol used on the traffic between two devices. It is expected to have only the ModBus/TCP inside.
num_pack	Number of Packets that compose the flow, created on the Feature Generation step, represents the count of number of packets on a flow.
size_pack	Size of Packets, created on the Feature Generation step, represents the sum of the size of all packets that compose the flow.
ppf	Packets per flow, created on the Feature Generation step, represents the count of all packets that compose the flow.
bpf	Bytes per Flow, created on the Feature Generation step, represents the ratio between size_pack and flow_duration.
avg_size_pack	Average of packets size, created on the Feature Generation step, represents the average size of the packets that compose the flow.
flow_duration	Flow Duration, created on the Feature Generation step, represents the duration of the flow.

4. Evaluation

In this section of the paper, we use a publicly available dataset, created by [Lemay and Fernandez 2016], to evaluate our approach. The evaluation was accomplished as follows. First, the dataset generated by Lemay and Fernandez [2016] was used as input for the Flow Generation step. Then, the IP flows extracted in the previous step were sent to the Feature Creation step. Finally, in the Supervised Classification step, we employed nine machine learning algorithms on the extracted features.

This section provides, in the first place, a description of the dataset, its testbed, and the methodology. Then, we present the evaluation metrics and the results using each algorithm.

4.1. Dataset

The main reason that discourages companies from sharing their datasets with the scientific community is the sensitivity of the data that travel on SCADA networks. Yussof et al. [2014] discussed this issue in their work. Therefore, there are few datasets available publicly, and the few that are available are generated in controlled environments or have limitations on data confidentiality.

Aware of the limitations of the few datasets available, Lemay and Fernandez [2016] developed in a controlled environment a labeled dataset for SCADA networks using the Modbus protocol, making it public for studies and evaluations in intrusion detection area. The dataset, which was used in this paper, includes about 6 hours of both malicious and non-malicious traffic on a simulated industrial network. For further information about the workload and the network, please check Lemay and Fernandez [2016].

Although most of the traffic in the dataset is related to automated transactions between masters and slaves, the few discrepancies in time patterns or packet size can represent a human interacting with the system or also, an attack. The dataset includes attacks such as running a remote exploit, uploading files to the infected network, fingerprinting attack, and sending an unauthorized command to a controller.

The *pcap* files provided by the original dataset were then processed by the Network Gathering module, and finally aggregated into flows by the Flow Generation Module. It is important to mention that only the dataset files with malicious activity were used, meaning that only the files named Moving_two_files_Modbus_6RTU.pcap, Send_a_fake_command_Modbus_6RTU_with_operate.pcap, CnC_uploading_exe_modbus_6RTU_with_operate.pcap and 6RTU_with_operate.pcap were imported. The description of each file can be seen in [Lemay and Fernandez 2016].

The results of the conversion of the original dataset to a flow-based dataset are presented in Table 2.

Table 2. Original and flow-based datasets.

	Original Dataset (packets)	**Flow-Based Dataset (flows)**
Majority Class (non malicious)	16362	5488
Minority Class (malicious)	1405	38
Unbalance Ratio	11:1	144:1
Total	**17767**	**5526**

4.2. Test Environment and Methodology

In the Supervised Classification step, we tested nine two-class algorithms as follows: Support Vector Machine, Locally Deep Support Vector Machine, Averaged Perceptron, Neural Network, Logistic Regression, Bayes Point Machine, Boosted Decision Tree, Decision Forest, and Decision Jungle.

In this work, we aimed to use the most relevant algorithms from each classifiers family. From the family of discriminative classifiers, we selected Support Vector Machine (SVM), Locally Deep Support Vector Machine (LD-SVM), Averaged Perceptron and Neural Network. From the family of decision tree classifiers, we picked Boosted Decision Tree, Decision Forest, and Decision Jungle. At last, from the statistical classifiers family, Logistic Regression and Bayes Point Machine were chosen.

Support Vector Machine (SVM) and Locally Deep Support Vector Machine (LD-SVM) are ML algorithms that aim to predict the correct class by building a multidimensional vector, in which each instance of the dataset is classified. The goal is to obtain the largest margin of separation of the hyper-plane between classes. The main difference between the regular SVM and the LD-SVM is that the last one uses a different kernel that works better on large datasets.

Averaged Perceptron and Neural Network are ML algorithms that belong to the Neural Network family. Averaged Perceptron is the simplest type of Neural Network algorithm and uses a single neuron. It linearly classifies data using an activation function, in which the weighted data is used as input. The Neural Network algorithm is the gen-

eralization of Averaged Perceptron, which, instead of using a single neuron, has a set of neurons interconnected in a directed, weighted, and acyclic graph.

Logistic Regression is an ML algorithm that predicts the statistical probability of a sample belonging to a class using a logistic distribution of the samples. Bayes Point Machine is also a ML algorithm, which uses a Bayesian linear approach to classify data.

Boosted Decision Tree, Decision Forest, and Decision Jungle are ML algorithms based on decision tree classification. The Boosted Decision Tree is an ensemble learning method, in which each tree corrects the errors of its predecessor, and the prediction is made by evaluating the entire ensemble. Decision Forest, also known as Random Decision Forest, is an ensemble algorithm that classifies data by randomly constructing multiple decision trees and voting in whichever provides the closest probability of correct label. Decision Jungle is considered an extension of Decision Forest, which instead of using one path to every node, uses directed acyclic graphs to allow multiple paths from the root to each leaf.

The classifier module and all algorithms were implemented on Microsoft Azure Machine Learning Studio [Microsoft 2017]. Microsoft Azure Machine Learning Studio is a cloud predictive analytic tool that allows the implementation of many machine learning models and executes them in parallel. We executed all the algorithms using their default parameters. The algorithms tested and their defaults parameters can be seen in Table 3.

Table 3. Algorithm parameters.

Algorithm	Parameters
Support Vector Machine	**Create trainer mode:** Single Parameter **Number of iterations:** 1 **Lambda:** 0.001 **Normalize features:** True **Project to the unit-sphere:** False **Allow unknown categorical levels:** True
Locally-Deep Support Vector Machine	**Create trainer mode:** Single Parameter **Depth of the tree:** 3 **Lambda W:** 0.1 **Lambda Theta:** 0.01 **Lambda Theta Prime:** 0.01 **Sigmoid sharpness:** 1.0 **Number of iterations:** 15000 **Feature normalizer:** Min-Max normalizer **Allow unknown categorical levels:** True
Averaged Perceptron	**Create trainer mode:** Single Parameter **Learning rate:** 1.0 **Maximum number of iterations:** 10 **Allow unknown categorical levels:** True

Neural Network	**Create trainer mode:** Single Parameter **Hidden layer specification:** Fully-connected case **Number of hidden nodes:** 100 **Learning rate:** 0.1 **Number of learning iterations:** 100 **The initial learning weights:** 0.1 **The momentum:** 0 **The type of normalizer:** Min-Max normalizer **Shuffle examples:** True **Allow unknown categorical levels:** True
Logistic Regression	**Create trainer mode:** Single Parameter **Optimization tolerance:** 10^{-7} **L1 regularization weight:** 1 **L2 regularization weight:** 1 **Memory size for L-BFGS:** 20 **Allow unknown categorical levels:** True
Bayes Point Machine	**Number of training iterations:** 30 **Include bias:** True **Allow unknown values in categorical features:** True
Boosted Decision Tree	**Create trainer mode:** Single Parameter **Maximum number of leaves per tree:** 20 **Minimum number of samples per leaf node:** 10 **Learning rate:** 0.2 **Number of trees constructed:** 100 **Allow unknown categorical levels:** True
Decision Forest	**Resampling method:** Bagging **Create trainer mode:** Single Parameter **Number of decision trees:** 8 **Maximum depth of the decision trees:** 32 **Number of random splits per node:** 128 **Minimum number of samples per leaf node:** 1 **Allow unknown values for categorical features:** True
Decision Jungle	**Resampling method:** Bagging **Create trainer mode:** Single Parameter **Number of decision DAGs:** 8 **Maximum depth of the decision DAGs:** 32 **Maximum width of the decision DAGs:** 128 **Number of optimization steps per decision DAG layer:** 2048 **Allow unknown values for categorical features:** True

To evaluate the different algorithms, we used the hold-out approach. According to the hold-out approach, the dataset was randomly divided into two subsets. The first subset was called training set. It contained 60% of the original data and was used to build the predictive model. The second subset was called test set. It contained the remaining 40% of the original data and was used to test and measure the performance of the model on unseen inputs.

In addition to the hold-out approach, we also applied a downsampling on the majority class of the training set. Downsampling is a technique that removes some data from a given dataset randomly. The objective of this technique is to balance the classes on the training set. According to He and Ma [2013], some ML algorithms perform better on anomaly detection when the imbalance between classes is reduced. Therefore, we tested each algorithm reducing the imbalance as presented in Table 4, where the values 1%, 2%, 3%, 4% and 5% mean the reduction rate in the training dataset regarding the 100% value. For example, a reduction to 5% will reduce from 3293 to 132 samples. Also, we made the same tests without the downsampling to compare results.

When detecting malicious traffic, we usually deal with imbalanced classes, since most of the traffic is non-malicious rather than malicious. In the original dataset, handled in the Network Gathering step, we dealt with an 11:1 ratio between malicious and non-malicious packets. After the aggregation into a flow based dataset, this ratio increased to 144:1 between malicious and non-malicious flows, rising the importance of studying the impact of downsampling on the overall results.

Table 4. Downsampling values and imbalance ratio

	Training Dataset						**Test Dataset**	**Total**
	1%	**2%**	**3%**	**4%**	**5%**	**100%**		
Non-malicious flows	33	66	99	132	165	3293	2195	**5488**
Malicious flows	23	23	23	23	23	23	15	**38**
Ratio	1:1	3:1	4:1	5:1	7:1	143:1	146:1	**144:1**

4.3. Evaluation Metrics and Results

He and Ma [2013] recommend the use of F1-Score and Averaged Accuracy as the main metrics to evaluate the performance of any ML algorithm with an imbalanced dataset. We also added Accuracy for further reference. According to He and Ma [2013], a dataset is considered imbalanced if the imbalance between classes has a higher ratio than 10:1. In our case, the total ratio is 144:1, as previously presented in Table 2. In the training set, the downsampling technique reduced that imbalance from 144:1 up to a 1:1 ratio, although it represents heavy losses in information due to the random characteristic of the technique. According to the authors, the biggest problem of class imbalance is the increase of error rate as the imbalance increases, which impacts directly on the ability to learn the minority class, biasing the major class. That means that not every algorithm will perform within an acceptable level in this situation. The evaluation metrics, their formulas, scores, and a brief description are presented next.

1. $F1\_Score = 2\frac{Precision*Recall}{Precision+Recall} = \frac{2TP}{2TP+FP+FN}$
   It measures the harmonic mean of Precision and Recall. Precision stands for the ratio between correct predictions (TP) and the number of all correct (TP) and incorrect predictions (FP). Recall stands for the ratio between correct predictions (TP) and the number of all correct predictions (TP + FN). F1 Score is represented as a continuous number between 0 and 1, being 1 as the best value and 0 the worst.
2. $Averaged\_Accuracy = \frac{Recall+Specifity}{2} = \frac{TP}{2TP+2FN} + \frac{TN}{2TN+2FP}$
   It measures the balanced accuracy, balancing the scores between Recall and Specifity. Specifity stands for the ratio between correct predictions on the majority class

(TN) and the number of all correct predictions on the majority class (TN + FP). Averaged Accuracy is represented as a continuous number between 0 and 1, being 1 as the best value and 0 the worst.

3. $Accuracy = \frac{TP+TN}{TP+FN+FP+TN}$
   It measures the closest an implemented model correctly predicts all cases. It is represented as a continuous number between 0 and 1, being 1 as the best value and 0 the worst

TP, TN, FP, and FN stand for true positives, true negatives, false positives and false negatives respectively.

On the execution of each classification technique, we ran each algorithm ten times for each downsampling value and also for the non-downsampled training dataset. After each execution, each one of the nine algorithms generated 100 values corresponding to the thresholds evaluated. After all executions, we collected 54000 values as results and their averages are presented in Table 5, where the corresponding downsampling ratios are presented between brackets.

Table 5. Results for evaluation metrics regarding the nine algorithms

	F1-Score	**Average Accuracy**	**Accuracy**
Support Vector Machine	0.1278 (5%)	0.5877 (2%)	0.9934 (100%)
Locally Deep Support Vector Machine	0.7523 (100%)	0.9489 (5%)	0.9970 (100%)
Averaged Perceptron	0.4455 (5%)	0.8114 (1%)	0.9942 (100%)
Neural Network	0.5399 100%	0.8597 (3; 4%)	0.9935 (100%)
Logistic Regression	0.4416 (100%)	0.8744 (3; 4%)	0.9946 (100%)
Bayes Point Machine	0.7730 (100%)	0.9518 (100%)	0.9970 (100%)
Boosted Decision Tree	0.8467 (100%)	0.9684 (4; 5; 100%)	0.9977 (100%)
Decision Forest	0.8026 (100%)	0.9694 (100%)	0.9972 (100%)
Decision Jungle	0.8508 (100%)	0.9690 (4; 100%)	0.9979 (100%)

Analyzing the results presented in Table 5, the five best algorithms were Decision Jungle, Boosted Decision Tree, Decision Forest, Bayes Point Machine and Locally Deep Support Vector Machine. Further discussion on those results will be carried out in the Discussion subsection.

Another two evaluators were also implemented to compare the results of the five best algorithms. The first, presented in Figure 2, is the Receiver Operating Characteristic (ROC) curve, allowing to visually see the variation of the True Positive Rate versus the False Positive Rate. The second evaluator is presented in Figure 3, showing the trade-off between Precision and Recall.

In Figure 2, we present the ROC curve for the five best algorithms, each of them showing their best results without the use of the downsampling technique. In those graphs, we can see that Decision Jungle and Decision Forest have the highest and almost identical performance, followed by Bayes Point Machine.

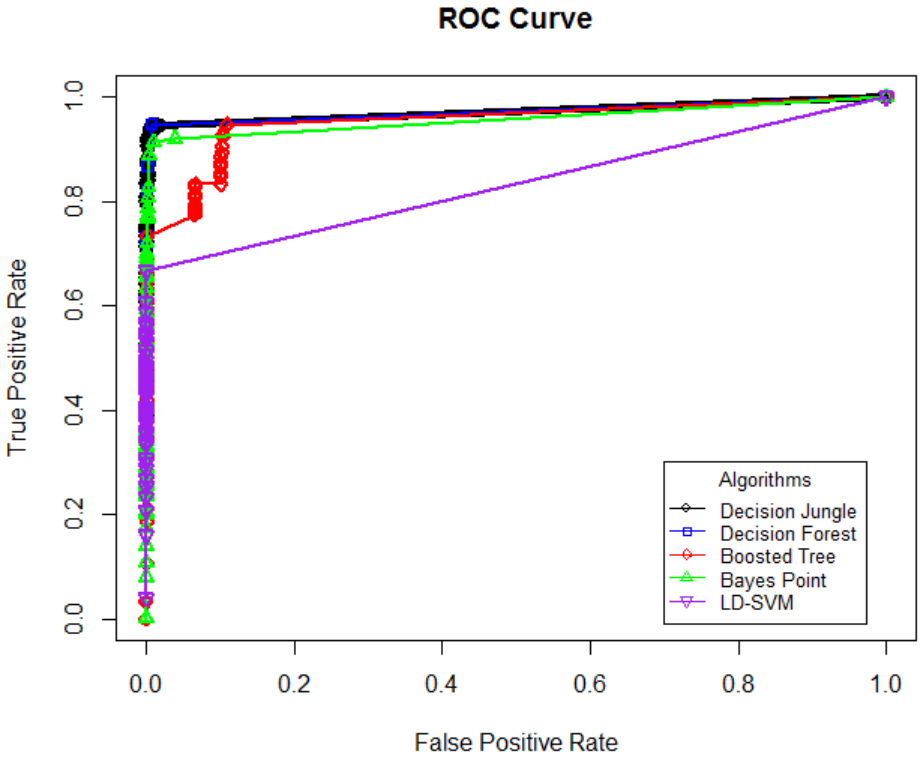

Figure 2. ROC Curve

In Figure 3, we present the Precision-Recall curve for the five best algorithms, each of them showing their best results without the use of the downsampling technique. In those graphs, we can see that Decision Jungle has the highest performance followed by Boosted Tree.

4.4. Discussion

The results presented in Table 5, and the results shown in Figure 2 and Figure 3 allow us to reach some conclusions to be discussed in this section.

First, based on F1-Score and Average Accuracy presented in Table 5, we can conclude that all three Decision Tree algorithms outperformed the other ones, presenting the best of all F1-Score over 80% and Average Accuracy over 96%. The only other two algorithms that showed similar, although inferior, results are Bayes Point Machine and Locally Deep Support Vector Machine.

Second, analyzing the overall results presented in Table 5, we conclude that the downsampling technique applied on the training dataset did not provide significant aid to the best-scored algorithms. In fact, the downsampling helped only the Averaged Perceptron and the Support Vector Machine, but not enough to make them relevant toward the construction of an IDS.

Figure 2 shows the ROC curve of the five best algorithms selected through the results presented in Table 5. The ROC curve allowed us to compare the trade-off between

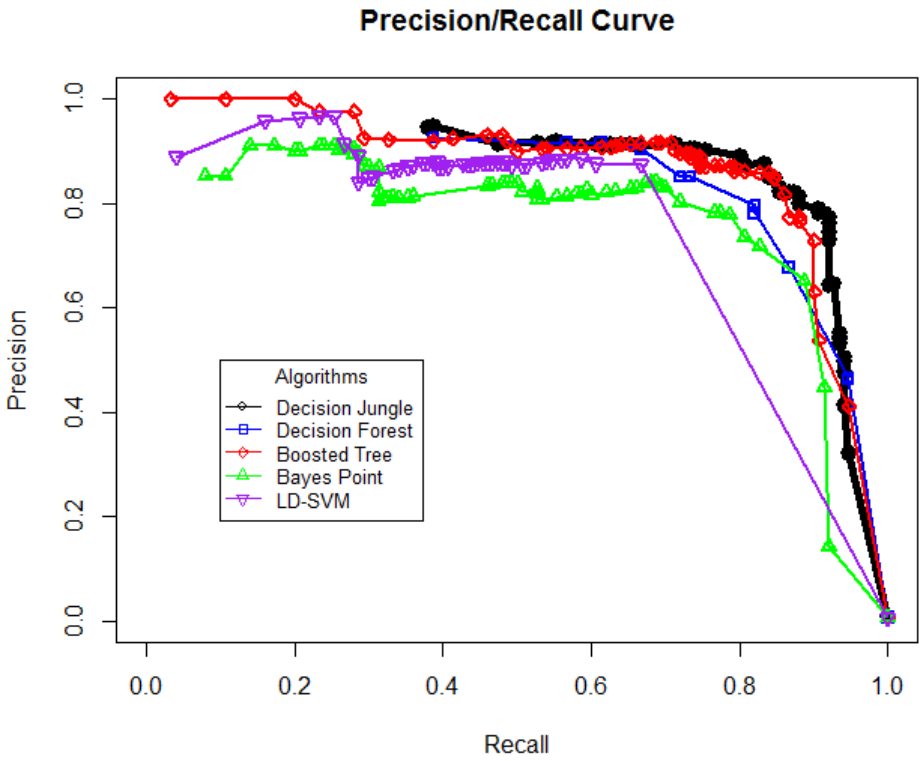

Figure 3. The Precision Recall Curve

the True Positive Rate (TPR) and the False Positive Rate (FPR) of the algorithms. In ROC curves, the better results are in the upper left corner, where the TPR is closer to 1 and the FPR is closer to 0. In Figure 2, we could see that the algorithms Decision Jungle and Decision Forest sustained a higher performance through all the threshold variation when compared to the other algorithms. These two algorithms reached, for the most of the graph, the desired top left corner. On the other hand, Boosted Decision Tree could not sustain similar results. Here, it is noteworthy that Bayes Point Machine did not outperform the two best algorithms, but had an overall good performance. Locally Deep SVM had the worst results on this evaluator.

Figure 3 shows the Precision-Recall Curve of the same five best algorithms selected through the results presented in Table 5. The Precision-Recall Curve is a type of curve that visually shows the quality of the results of any given classifier. A classifier with high recall but low precision will detect the most of the attacks, but will generate many false positives. A classifier with low recall but high precision will generate few false positives, but will miss many attacks. The desired output is reached on the upper right corner, where there is a high Precision with a high Recall.

In the presented results, we could see that the Decision Jungle algorithm presented higher overall results than any other one. The next best result was reached by Boosted Tree algorithm. Here, we could also see that although Decision Forest presented high results in Table 5 and in the ROC Curve, it did not have a good precision-recall trade-off. All other algorithms did not perform well enough on this evaluation.

After analyzing the results, we observed that the Decision Jungle algorithm presented the highest scores in Table 5 and an overall good performance in the ROC Curve and Precision/Recall Curve. Therefore, we pointed out it as the best solution among the

algorithms we evaluated.

We believe that there are two main reasons for the good performance of Decision Jungle. The first one is the ensemble learning. It employs multiple learning models, leveraging the classification power of multiple base learners, such as decision trees, to improve the classification performance over traditional algorithms.

In the case of Decision Jungle [Shotton et al. 2013], multiple decision trees are connected using directed acyclic graphs, allowing multiple paths from the root to each leaf. That type of connection among nodes allowed a higher degree of merging between leafs, reducing the complexity of the classifier and the number of misclassifications. For Modbus/TCP traffic, this approach presented a significant leverage in comparison with other non-ensemble algorithms.

The second reason that helped the Decision Jungle to be the best among the others is its awareness of class imbalance. That awareness is due to ensemble model. In our aggregation, in Flow Generation step, we ended with a class imbalance of 144:1. That imbalance could have impacted the poor results of some algorithms like SVM or Perceptron, but it did not impact more advanced models like the Decision Tree family algorithms. We believe that this is directly related to the ensemble model. We also noticed that the downsampling had no benefit on the Decision Jungle algorithm, since its best results were reached with no downsampling.

5. Conclusion

In this paper, we evaluated the performance of nine different ML algorithms in classifying IP flows of an SCADA ModBus/TCP network, analyzing the impact of class imbalance in the results. Those algorithms may be the core of the detection engine of an Intrusion Detection System (IDS). The role of an IDS is to monitor the network, collecting data to notify the administrators.

In a SCADA ModBus/TCP network, the traffic presented is different from a regular TCP/IP network. The ModBus/TCP network is a restricted network, where the addition of a new device is rare. The traffic is standardized and predictable between masters and slaves due the polling process. This characteristic allowed us to arrange similar packets into IP flows. In the flows, we also compute the average packet size, count number of packets and the flow duration. The predictability in the SCADA traffic also allowed us to establish a communication pattern that could be analyzed using a machine learning supervised classifier.

The analysis of the results of the studied algorithms showed us that the Decision Jungle presented the best scores for the metrics results (F1 Score and Average Accuracy) and the overall performance in the ROC Curve and Precision/Recall Curve. The second best performance was reached by Boosted Tree algorithm, presenting a good score for the evaluation metrics, almost the same performance that Decision Jungle in the ROC Curve and a standard performance in the Precision/Recall Curve. We believe that those results were accomplished due to the ensemble model in the construction of those algorithms, allowing them to outperform all other simpler algorithms.

As future work, we intend to extend this work by using semi-supervised ML algorithms. That will allow us to reduce the dependency of the labeled data in the dataset. We also intend to use differents settings on Microsoft Azure parameters to improve the detection rate of all algorithms.

References

Claise, B., Trammell, B., and Aitken, P. (2013). Specification of the IP Flow Information Export (IPFIX) Protocol for the exchange of flow information (2013).

He, H. and Ma, Y. (2013). *Imbalanced learning: foundations, algorithms, and applications*. John Wiley & Sons.

Junejo, K. N. and Goh, J. (2016). Behaviour-based attack detection and classification in cyber physical systems using machine learning. In *Proceedings of the 2nd ACM International Workshop on Cyber-Physical System Security*, pages 34–43. ACM.

Lemay, A. and Fernandez, J. M. (2016). Providing SCADA network data sets for intrusion detection research. *9th USENIX Workshop on Cyber Security Experimentation and Test (CSET '16)*, pages 1–8.

Linda, O., Vollmer, T., and Manic, M. (2009). Neural network based intrusion detection system for critical infrastructures. In *Neural Networks, 2009. IJCNN 2009. International Joint Conference on*, pages 1827–1834. IEEE.

Loukas, G. (2015). *Cyber-physical attacks: A growing invisible threat*. Butterworth-Heinemann.

Microsoft (2017). How to choose algorithms for Microsoft Azure machine learning.

Miller, B. and Rowe, D. (2012). A survey SCADA of and critical infrastructure incidents. In *Proceedings of the 1st Annual conference on Research in information technology*, pages 51–56. ACM.

Ntalampiras, S. (2015). Detection of integrity attacks in cyber-physical critical infrastructures using ensemble modeling. *IEEE Transactions on Industrial Informatics*, 11(1):104–111.

Piggin, R. (2015). Are industrial control systems ready for the cloud? *International Journal of Critical Infrastructure Protection*, 9(C):38–40.

Schuster, F., Paul, A., Rietz, R., and König, H. (2015). Potentials of using one-class SVM for detecting protocol-specific anomalies in industrial networks. In *Computational Intelligence, 2015 IEEE Symposium Series on*, pages 83–90. IEEE.

Shotton, J., Sharp, T., Kohli, P., Nowozin, S., Winn, J., and Criminisi, A. (2013). Decision jungles: Compact and rich models for classification. In *Proc. NIPS*.

Swales, A. et al. (1999). Open ModBus/TCP specification. *Schneider Electric*, 29.

Yang, Y., McLaughlin, K., Littler, T., Sezer, S., Pranggono, B., and Wang, H. (2013). Intrusion detection system for IEC 60870-5-104 based SCADA networks. In *Power and Energy Society General Meeting (PES), 2013 IEEE*, pages 1–5. IEEE.

Yusheng, W., Kefeng, F., Yingxu, L., Zenghui, L., Ruikang, Z., Xiangzhen, Y., and Lin, L. (2017). Intrusion detection of industrial control system based on Modbus TCP protocol. In *Autonomous Decentralized System (ISADS), 2017 IEEE 13th International Symposium on*, pages 156–162. IEEE.

Yussof, S., Rusli, M. E., Yusoff, Y., Ismail, R., and Ghapar, A. A. (2014). Financial impacts of smart meter security and privacy breach. In *Information Technology and Multimedia (ICIMU), 2014 International Conference on*, pages 11–14. IEEE.

