
# IoT4CPS – Trustworthy IoT for CPS

FFG - ICT of the Future
Project No. 863129

# Deliverable D4.3
# Analytical Toolbox

The IoT4CPS Consortium:
AIT – Austrian Institute of Technology GmbH
AVL – AVL List GmbH
DUK – Donau-Universität Krems
IFAT – Infineon Technologies Austria AG
JKU – JK Universität Linz / Institute for Pervasive Computing
JR – Joanneum Research Forschungsgesellschaft mbH
NOKIA – Nokia Solutions and Networks Österreich GmbH
NXP – NXP Semiconductors Austria GmbH
SBA – SBA Research GmbH
SRFG – Salzburg Research Forschungsgesellschaft
SCCH – Software Competence Center Hagenberg GmbH
SAGÖ – Siemens AG Österreich
TTTech – TTTech Computertechnik AG
IAIK – TU Graz / Institute for Applied Information Processing and Communications
ITI – TU Graz / Institute for Technical Informatics
TUW – TU Wien / Institute of Computer Engineering
XNET – X-Net Services GmbH



*For more information on this document or the IoT4CPS project, please contact:*
Mario Drobics, AIT Austrian Institute of Technology, mario.drobics@ait.ac.at

## Document Control

| | |
|---|---|
| **Title:** | Analytical Toolbox |
| **Type:** | Public |
| **Editor(s):** | Christian Lettner |
| **E-mail:** | christian.lettner@scch.at |
| **Author(s):** | Arndt Bonitz, Faiq Khalid, Christian Lettner, Muhammad Shafique, Radoslava Svihrova, Patrick Traxler, Heribert Vallant |
| **Doc ID:** | D4.3 |

## Amendment History

| Version | Date | Author | Description/Comments |
|---|---|---|---|
| V0.1 | 13.05.2019 | Patrick Traxler | Initial version: structure, introduction |
| V0.2 | 16.05.2019 | Arndt Bonitz | Anomaly Detection in Operating Systems |
| V0.3 | 23.05.2019 | Patrick Traxler | References |
| V0.4 | 30.05.2019 | Faiq Khalid | Anomaly detection in Hardware |
| V1.0 | 31.05.2019 | Patrick Traxler | Editing, executive summary |
| V1.01 | 03.06.2019 | Branka Stojanovic, Heribert Vallant | Document review |
| V1.02 | 14.06.2019 | Patrick Traxler | Integrating comments |
| V1.03 | 17.6.2019 | Faiq Khalid | Integrating the comments and adding more details |
| V1.031 | 01.07.2019 | Heribert Vallant | Fix some minor issues which were introduced during the integration of comments |
| V2.00 | 16.3.2020 | Christian Lettner | Updated Chapter 2, Analytical toolbox |
| V2.01 | 21.03.2020 | Arndt Bonitz | Updated Chapter 3, concept of integration |
| V2.02 | 25.3.2020 | Radoslava Svihrova, | Updated Chapter 2, ML based methods |
| V2.03 | 19.04.2020 | Faiq Khalid | Updated Chapter 4, Anomaly detection in Hardware (MacLeR) |
| V2.04 | 10.05.2020 | Albert Treytl, Branka Stojanovic | Review of document |
| V2.05 | 22.05.2020 | Faiq Khalid | Addressed the reviewers' comments |
| V2.06 | 25.05.2020 | Arndt Bonitz | Integrating comments |
| V2.07 | 25.05.2020 | Christian Lettner | Integrating comments |



The IoT4CPS project is partially funded by the "ICT of the Future" Program of the FFG and the BMVIT.

bmvit Bundesministerium für Verkehr, Innovation und Technologie

FFG Österreichische Forschungsförderungsgesellschaft

## Content

## Abbreviations

| | |
|---|---|
| ÆCID | Automatic Event Correlation for Incident Detection |
| ACC | Accuracy |
| ADC | Analog to Digital Converter |
| AE | Autoencoder |
| BACC | Balanced Accuracy |
| BCE | Binary Cross-Entropy |
| CAB | Current Acquisition Block |
| CNN | Convolutional Neuronal Networks |
| CPS | Cyber-physical system |
| CE | Cross-Entropy |
| DSE | Design Space Exploration |
| DT | Decision Tree |
| FAR | False Alarm Rate |
| FNN | Feed Forward Neural Network |
| FN | False-negative |
| FP | False-positive |
| IC | Integrated Circuit |
| ICT | Information and communications technology |
| IDS | Intrusion Detection System |
| IoT | Internet of Things |
| IP | Internet Protocol |
| IT | Information Technology |
| GMM | Gaussian Mixture Model |
| HIDS | Host IDS |
| HT | Hardware Trojans |
| LASSO | Least Absolute Shrinkage and Selection Operator |
| LR | Linear Regression |
| MAE | Mean Absolute Error |
| ML | Machine Learning |
| MLP | Multi-Layer Percepteron |
| MQ | Message Queue |
| MQTT | MQ Telemetry Transport |
| MSE | Mean Squared Error |
| MTU | Master Terminal Unit |
| NPV | Negative Predictive Value |
| OT | Operational Technology |
| PPV | Positive Predictive Value |
| SCADA | Supervisory control and data acquisition |
| SM | Softmax |
| TNR | True Negative Rate |
| TPR | True Positive Rate |
| PAC | Probably Approximately Correct |
| PC | Personal Computer |
| PSL | Property Specification Language |
| PV | Process Variations |
| RBF | Radial Basis Function |

| | |
|---|---|
| REST API | Representational State Transfer Application Interface |
| RF | Random Forest |
| RTU | Remote Terminal Unit |
| SELU | Scaled Exponential Linear Unit |
| SoC | System on a Chip |
| SP-CAB | Single Power-port Current Acquisition Block |
| SVM | Support Vector Machine |
| WC | Worst-Case |
| BC | Best-Case |
| 3PIP | 3rd Party Intellectual Property |

## Executive Summary

This deliverable describes methods for anomaly detection for IoT4CPS and its prototypical implementation as an Analytical Toolbox. Results have been achieved by the IoT4CPS partners AIT, SCCH, and TU Vienna during the first and second phase of the project. AIT developed a path within the ÆCID framework for future integration into the Analytical Toolbox and the IoT4CPS demonstrators. First steps for the integration of a message queue into ÆCID, have been undertaken and successfully tested. ÆCID stands for Automatic Event Correlation for Incident Detection and is an intelligent cybersecurity tool that uses special mathematical calculations to distinguish abnormalities from normal behavior. ÆCID is continuously developed by AIT. Whereas the work of AIT concerns mostly the operating system level, SCCH worked on machine learning-based anomaly detection on the network communication level. Supervised as well as semi-supervised methods are evaluated on public datasets for intrusion and attack (anomaly) detection. In addition, a prototypical implementation of the Analytical Toolbox that allows to perform online learning of machine learning models has been developed. The work of AIT and SCCH is complemented by contributions of TU Vienna, which mostly concerns anomaly detection in hardware using fine-grained power profiling of embedded microcontrollers and microprocessors.

# 1. Introduction

## 1.1 Analytical Toolbox for Anomaly Detection

Observing complex cyber-physical systems (CPS) or even networks of complex cyber-physical systems is a critical contemporary challenge for companies across Austria. The Internet of Things (IoT) offers basic technologies for collecting large amounts of data from CPSs around the world. Understanding and inferencing from this data is a core challenge for data analysis and knowledge engineering. For the purpose of improving the security of networks of CPSs, statistical inferencing, and knowledge reasoning from these massive amounts of data becomes exceptionally challenging. Attacks and intrusions are often not known to security experts. Thus, defining patterns in data for algorithmic detection of attacks and intrusions is not feasible. Even if researchers and security experts identify some of these patterns, the attackers will move on to other or improved approaches and tools. It is thus clear that companies are seeking for software tools to support them in their task to monitor and understand the risk and security threats within their networks. Such networks may comprise thousands to millions of systems. The Analytical Toolbox for Anomaly Detection aims at supporting companies in their task to keep their systems and networks safe and secure.

Anomaly detection can be performed with the help of machine learning, that tries to identify atypical system behaviour. It is similar to outlier detection in statistics. The general approach is to train models of the typical system behaviour and then to identify deviations from it. These deviations are the anomalies. It is of importance to note that this approach does not assume any knowledge about the attacks, tools for attacking, or about the attackers. In terms of machine learning, it is a semi-supervised approach. We also note that besides training the models, manual modelling of the typical system behaviour by experts may be viable.

The use cases of IoT4CPS suggests several improvements of general anomaly detection due to the restriction to networks of cyber-physical systems. Such systems and networks have their own characteristics. We distinguish between three levels of data sources for anomaly detection in the IoT4CPS project: Data from hardware, from operating systems, and from the network traffic. These three levels of data sources are studied by the three project partners TUW, AIT and SCCH, respectively.

- Network traffic (SCCH): The data source is mostly TCP/IP information. This includes the IP addresses, TCP connections, size of packets, etc. For networks of CPS, this network communication data may be very homogenous. For example, data is sent every day at the same time from a CPS. A deep packet insepction, i.e. inspecting the actual data sent over the network, is not performed.
- Operating system (AIT): The data source is comprised mostly of system logs and application events. This includes information about logins, executed programs, file system operations and so on. Logs can be acquired from IoT or CPS devices, as well as from backend servers.
- Hardware (TUW): The power consumption of particular hardware components together with physical measurements at the hardware level is the data pool for this type of analysis. An example of a data source is the power consumption of the device recorded by some sensors.

These three rather different types of data sources are the input for anomaly detection algorithms. Anomaly detection algorithms output the description of an anomaly which is usually defined as an event. Events have a start time, end time and perhaps an associated value. For the purpose of security monitoring, an estimation of the risk of the anomaly is preferable. In the practice of monitoring tools, anomalies are often not directly presented to the user of the monitoring tool. The reason is that anomalies may occur too frequently. Instead, only the most critical or aggregated anomalies are presented to the user as alarms. Alarms are associated with a text message describing the incident and may need to be confirmed after notification. Another usage of anomaly detection is to develop forensic tools for the purpose of identifying attacks and intrusions in historical data.

The Analytical Toolbox serves as a demonstrator for the applicability of anomaly detection algorithms to increase security and safety for cyber-physical systems (CPS) and for the Internet of Things (IoT.)

## 1.2 Relation to Business Needs and Use Cases

Anomaly detection, as outlined above, relates to the usecases, i.e., automated driving (WP6) and smart production (WP7). Requirements are the availability of data. A precise description of the data models that are employed in this research can be found in the following sections.

Anomaly detection at all three levels (hardware, operating systems, network traffic) applies to the business needs of

- AVL: IoT for the Connected Vehicle (see D2.2., Chapter 3.3)
- IFAT: Trustworthy radio connectivity solutions in smart production use-cases (see D2.2., Chapter 3.4)
- NXP: Integrity and Authenticity Check of Complex Systems (see D2.2., Chapter 3.5)

Anomaly detection in hardware applies also to

- TTTech: Secure and Safe Platform for Automated Driving (see D2.2, Chapter 3.7)

Anomaly detection in software (operating systems, network traffic) applies to the use case of

- X-Net: Security by Isolation / Production of Storage Media for IoT devices (see D2.2, Chapter 3.8)

# 2. Anomaly Detection in Network Traffic

## 2.1 Public Datasets for Intrusion Detection Modelling

The dataset from the KDD Cup '99 [16] is widely used for the evaluation of models for attack and intrusion detection. The dataset consists of 41 features and each observation is labeled as either normal (no attack) or with the type of attack. There are four main categories for the simulated attacks:

- DoS (Denial of Service attack): denial-of-service, e.g., syn flood
- U2R (User to Root attack): unauthorized access to local superuser (root) privileges, e.g., various buffer overflow attacks
- R2L (Remote to Local attack): unauthorized access from a remote machine, e.g., guessing password
- Probe (Probing attack): surveillance and other probing attacks, e.g., port scanning

Based on this dataset, two sub-datasets were created: KDDTrain+ and KDDTest+, which split the dataset into a training and into a testing dataset, mainly to get reproducible and comparable results for model training and evaluation. The KDDTrain+ dataset contains 125973 and the KDDTest+ contains 22544 labeled samples. The KDDTest+ set is not from the same probability distribution as the KDDTrain+ set. The test set contains 17 novel attacks, which are not contained in the train set, while 2 attack types are not present in the test set. This makes the task more realistic as there are also novelties in test data, which needs to be handled. Section 2.2 uses the KDDTrain+ and KDDTest+ sets for evaluating the introduced methods.

Another publicly available dataset is the CIC-IDS-2017 dataset [12][14]. This dataset consists of network data in the duration of 5 working days. On Monday, there was only a normal (benign) network flow with no attacks. On Tuesday there are 2 brute force attacks: FTP-Patator and SSH-Patator. Both of them lasted 1 hour. On Wednesday there were some Denial-of-Service attacks in the morning and one Heartbleed attack in the afternoon. On Thursday web attacks occurred. On Friday there were some infiltration attacks, as well as some other attacks. The dataset is not balanced, there is approximately 80% of benign network traffic and only 20% of malicious network traffic.

## 2.2 Machine Learning-based Methods for Anomaly Detection

In this section, we introduce machine learning methods used for anomaly detection on network traffic. First, we briefly introduce supervised approaches. All supervised methods require data from negative classes (normal or benging network traffic) and positive classes (attack or malicious network traffic) for training. However, data of the positive class, i.e., attacks, are not always available, or only a small number of samples containing information about malicious behaviour is accessible. In addition, even if the training dataset contains some of the attacks, it is highly likely that future data will contain new types of attacks, which we call novelties. Since all supervised methods rely on the assumption that the future data will be from the same probability distribution as the data used for training the model, detecting novel attack types might be impossible.

The idea to overcome both of these problems (i.e., novelties in the future data and too small number or any samples with information about malicious behaviour in training dataset) is to use a method that requires observations only from one class. The method learns patterns for benign samples and issues a warning if a new pattern, i.e., an anomaly, occurs. For this purpose, we introduce a semi-supervised model for anomaly detection based on the reconstruction error from undercomplete autoencoders.

In the following, only a brief overview and evaluation of the applied methods will be provided. You can find detailed information in the accompanying master thesis [15] for this deliverable.

### 2.2.1 Supervised Methods

The supervised learning approaches rely on the statistical learning theory described in [18] and on the probably approximately correct (PAC) framework [19], which rely on the assumption that the training and testing set are from the same probability distribution. In each method, there are two types of parameters that have to be optimized in order to learn the model which predicts the binary labels of the future data as correctly as possible: the model parameters $w \in W$, which are obtained by learning mechanism, i.e. by minimization of the loss function of the model, and the hyperparameters $\theta \in \Theta$, which are selected by the Bayesian optimization procedure as described in [6].

#### *Tree-based methods*

Tree-based methods are non-parametric supervised methods widely used not only for binary classification tasks. A decision tree (DT) is conceptually very simple but powerful. The aim of the algorithm is to learn the hierarchy of if/else statements by splitting the feature space into Q disjoint regions. The Gini index is used in each split as a criterion for selecting the variables and thresholds for the split. The procedure stops if the stopping criteria being fulfilled, for example, the minimal number of samples in one node is reached. This node is called a terminal node or leave. After the decision tree is grown to its full size, it can be post-pruned (this is a form of regularization). The final decision for the label is based on the majority of votes in the terminal nodes. The DT method is a greedy algorithm, i.e. the decisions made in each split are locally optimal, however, might not be globally optimal. Hence, the disadvantage of the decision tree is that it can easily overfit the training data.

As a single decision tree can suffer from high variance and thus can easily overfit, an ensemble of DTs called random forest (RF) was introduced. The idea of RF is to use the bootstrap principle to introduce randomness when building an ensemble of weak classifiers, shortly known as bagging (bootstrap aggregation). The final classification is then based on the average of votes from the decision trees in the forest. All trees in the RF are fully grown, i.e. without post-pruning. In addition, to overcome the correlation of the trees, only a subset of variables is used for selecting each split when building the trees.

#### *Support Vector Machines*

Another supervised method for binary classification is the support vector machine (SVM) method. The goal of SVM is to find a separating hyperplane in the feature space of the training data such that it distinctly classifies the data points and minimize the empirical classification error. At the same time, it maximizes the margin between the samples from positive and negative classes. SVMs are thus also known as maximum margin classifiers.

The data is typically not linearly separable. In order to provide a non-linear decision boundary, the data is transformed into a Hilbert space and back by an inner product kernel function, in which the linear decision hyperplane with the maximum margin between the transformed data is found. We use the radial basis function (RBF) as the kernel function. The optimal decision boundary is then found by maximization of the Lagrange dual form, as the solution of the dual form requires only the known form of the kernel function instead of the unavailable functional form of the mapping function to Hilbert space. By introducing slack variables, the decision boundary is smooth, but some misclassifications are introduced.

#### *Logistic regression*

Logistic regression is the state-of-the-art binary classification method in statistics and used for binary classification of the observations produced by the computer network. The idea is to model the Bernoulli distributed response binary variable by a logistic function. For the estimation of the model parameters, typically the maximum likelihood method, i.e., maximization of the logarithm of the likelihood, is used. The minimization of the binary cross-entropy leads the same estimated regression coefficients. As the negative log-likelihood, or

equivalently the binary cross-entropy loss, might be monotone in some variables, the minimum is not possible to reach, and regression coefficients would be estimated as infinitely large. To overcome this problem the regularization by elastic net, i.e., the combination of LASSO penalty and Thikonov regularization term, is used.

#### *Feed-forward neural network*

By introducing a hidden structure to the logistic regression model a feed-forward neural network model is built. The idea is to introduce non-linearity to the final decision boundary of the parametric supervised learning method by adding hidden layers consisting of neurons with non-linear activation functions. The input to the hidden neuron is the weighted sum of the outputs of the neurons from the previous layer. The weights mentioned are the model parameters which are estimated by the backpropagation algorithm.

In order to overcome the vanishing gradient problem which typically occurs in too deep neural networks when the derivative of the activation function is bounded from above by some small number, e.g., derivation of the sigmoid is bounded by 0.25 and propagated many times, the self-normalizing neural network proposed in [8] will be used. The idea is to use the proposed SELU activation function together with the LeCun normal initialization of the model parameters and thus ensure the normality of the weights. Thanks to this self-normalizing property, the gradient does not vanish anymore during the training procedure. Note that for the binary classification task, it is still required to use the sigmoid activation function in the output neuron. For the learning of the network, i.e., estimation of the model parameters, the stochastic gradient descent using mini-batch learning with momentum term is used as an optimization procedure.

### 2.2.2 Semi-supervised Methods

As already discussed, the motivation for using this approach is that future data can consist of novel attacks that were not present in the training set, or the training data contains too few or none malicious samples. Supervised approaches are not able to overcome these problems. Semi-supervised methods require labelled samples for training, however only from negative classes, which are relatively easy to obtain.

The idea of the method is to learn normal behaviour by training an undercomplete autoencoder. An autoencoder is an unsupervised deep learning method aiming to predict its input with a restriction on the number of hidden neurons in the hidden layers, i.e. to be lower than the number of neurons in the input layer. Afterwards new samples are presented to the trained model and the reconstruction error is observed. Based on a defined threshold the sample is then classified as either positive or negative. It is expected, that negative samples have much lower reconstruction error than the positive ones. The algorithm for choosing the threshold will be introduced in the following.

#### *Autoencoder*

The autoencoder model aims to learn the representation of the normal traffic produced by the computer network, i.e., patterns in the benign data provided. A model consists of two functions: an encoder and a decoder. The forward step is thus composed of compression of the data from input space to the hidden representation (of dimension specified by the number of neurons in the bottleneck layer) and followed by the decoding of the hidden representation to the output space (same dimension as the input space). The output is then compared to the input using appropriate loss functions (based on the type of the variable).

In order to preserve the nature of the input variables in the outputs, the activation functions in the corresponding outputs and the loss functions for evaluation of the divergence of the predicted value and the target value are selected as follows:

- The binary input variables are predicted by logistic function and evaluated by binary cross-entropy (BCE) loss function.
- The categorical variables are one-hot encoded in the pre-processing stage and the corresponding output neurons use SoftMax (SM) activation function with cross-entropy (CE) loss function. Note that each categorical variable has its own set of output neurons.
- The continuous variables are transformed by min-max transformation to the interval between 0 and 1 and are predicted by output neurons with linear activation function and evaluated by mean absolute error (MAE) loss function. The choice of the MAE over mean squared error (MSE) is because MAE penalizes values on the (0,1) interval more than MSE.

The value of the loss function of the whole model is then calculated as the weighted sum of the sets of outputs: one weight for binary outputs, one weight for continuous outputs, and one weight for each set of categorical outputs. Note that the weights $\omega$ are hyperparameters of the model and thus have to be tuned. They are then normalized to sum up to 1. The described architecture is depicted in Figure 1.

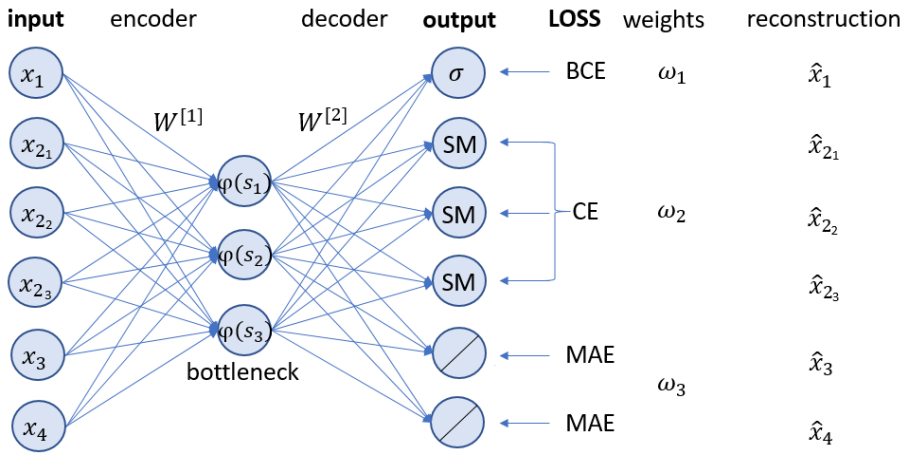



**Figure 1 – Autoencoder Architecture**

To enable the autoencoder to learn more complex structures extra hidden layers were introduced between the input and bottleneck layer. To preserve the symmetry of the model, the same number of layers with the same number of hidden neurons is mirrored and used also between the bottleneck and output layer.

The autoencoder is trained using the backpropagation algorithm. The selected activation function for the bottleneck layer is the hyperbolic tangent and for the other hidden layers the SELU activation function, which is also used in the supervised self-normalizing neural network model. The choice for the SELU activation function is not because of its self-normalizing property, as other requirements such as initialization and standardized inputs are not fulfilled anyway, but because of the good experimental results that were achieved. For weight updates, the Adam optimization procedure was used.

*Anomaly detection based on the reconstruction error*

This section describes how to set the optimal reconstruction threshold to classify samples as either positive or negative. The idea is to cluster the mean absolute differences between the output and the input layer of the autoencoder into two groups. The clustering is done based on the threshold identified by the point where two Gaussians with fixed means meet in the Mixture model. The algorithm is described in Listing 1.

**procedure** GMM_CLASSIF($AE, \{\boldsymbol{x}_n\}_{n=1}^{N}, \{\boldsymbol{x}_m\}_{m=N+1}^{N+M}$)

$\forall n : \hat{x}_n \leftarrow AE(\boldsymbol{x}_n)$

$\forall m : \hat{x}_m \leftarrow AE(\boldsymbol{x}_m)$

$\forall n : \zeta_n \leftarrow \frac{1}{D}\sum_{d=1}^{D} |\hat{x}_{nd} - x_{nd}|$

$\forall m : \zeta_m \leftarrow \frac{1}{D}\sum_{d=1}^{D} |\hat{x}_{md} - x_{md}|$

$\tau' \leftarrow q((\zeta_1, ..., \zeta_N), 0.9)$ ▷ $q(..., 0.9)$ is 0.9 quantile

Estimate: $\begin{cases} \hat{\mu}_0 = \frac{1}{\sum_m [\log(\zeta_m) < \tau']} \sum_m [\log(\zeta_m) < \tau'] \log(\zeta_m) \\ \hat{\mu}_1 = \frac{1}{\sum_m [\log(\zeta_m) \geq \tau']} \sum_m [\log(\zeta_m) \geq \tau'] \log(\zeta_m) \end{cases}$, for $\forall m$

Fit GMM with $\hat{\mu}_0$ and $\hat{\mu}_1$ fixed

$\tau \leftarrow$ point where Gaussian densities meet between $\hat{\mu}_0$ and $\hat{\mu}_1$

$\forall m : \hat{y}_m \leftarrow \begin{cases} 0 & \log(\zeta_m) < \tau \\ 1 & \log(\zeta_m) \geq \tau \end{cases}$

**return** $\hat{y}_m, m = N+1, ..., M$ ▷ Predicted labels

**Listing 1 – Anomaly detection based on reconstruction error**

AE represents the autoencoder trained on N benign samples denoted as $\{x_n\}_{n=1}^{N}$. The testing set $\{x_m\}_{m=N+1}^{N+M}$ consists of M samples from both positive and negative classes. The proposed algorithm uses only the information from benign samples of the training set, also for the estimation of the fixed means for the two Gaussian components. The algorithm then returns the predicted label for each sample based on the threshold $\tau$ obtained as the point where the two Gaussian components meet.

The proposed technique is then applied to the whole original training (including attacks) and testing dataset. The final mixture models can be found in Figure 2 and Figure 3 respectively. The left component, consisting of small values of reconstruction error, is assumed to contain benign samples and is denoted by blue colour, whereas the right component with higher values is assumed to contain malicious behaviour and is denoted by orange colour. The plots show the Gaussian Mixture Model (GMM) fitted to the logarithm of reconstruction errors. The final threshold $\tau$ used for the binary classification is on the right side of $\tau'$ (black line in the figure) in case of train data, in case of test data it is on the left side.

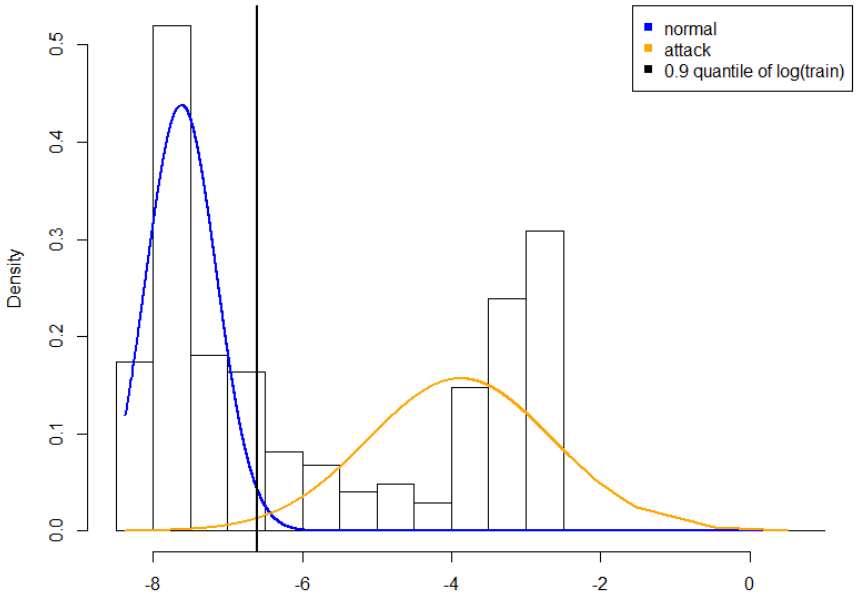



**Figure 2 – GMM fitted to the whole KDDTrain+ set**

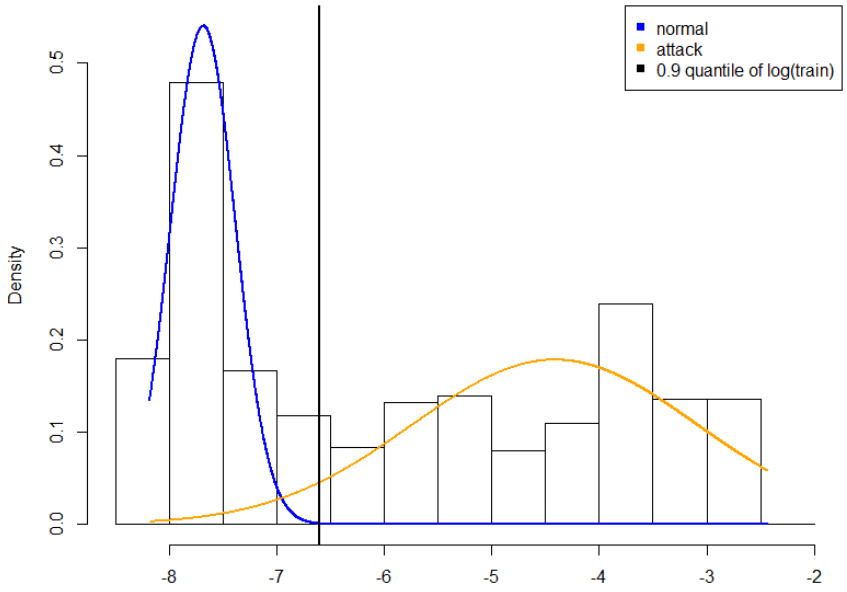



**Figure 3 – GMM fitted to the KDDTest+ set**

### 2.2.3 Evaluation

This section summarizes the evaluation results of the trained models using the predefined training dataset KDDTrain+ and testing dataset KDDTest+. The reason for choosing such setting is that it is publicly available, and since it is also used by other publications, the performance can be compared without bias towards a "better or worse split". The detailed results along with the hyperparameters selected by the Bayesian optimization procedure and afterwards used for the training of the final classifier can be found in the accompanying master thesis [15].

**Table 1 – Evaluation results on training data for all models**

| train | ACC % | BACC % | TPR % | TNR % | FAR % | NPV % | PPV % | $\kappa$ % |
|---|---|---|---|---|---|---|---|---|
| DT | 99.60 | 99.60 | 99.60 | 99.60 | 0.40 | 99.65 | 99.54 | 99.20 |
| RF | **99.99** | **99.99** | **99.98** | **99.99** | **0.01** | **99.99** | **99.99** | **99.98** |
| SVM | 99.82 | 99.82 | 99.76 | 99.87 | 0.13 | 99.80 | 99.85 | 99.64 |
| LR | 97.21 | 97.11 | 95.70 | 98.52 | 1.48 | 96.34 | 98.26 | 94.38 |
| FNN | 99.80 | 99.81 | 99.89 | 99.73 | 0.27 | 99.90 | 99.69 | 99.61 |
| AE+GMM | 92.09 | 92.10 | 92.26 | 91.95 | 8.05 | 93.17 | 90.89 | 84.12 |

Table 1 shows the evaluation results based on the training dataset only for all investigated models. Note that all reported values are in percentages. Based on the results obtained on the full training set we can see that RF model performed best for all measures. As the training set is approximately balanced (artificially, this is not expected in the real-world setting) and the cost for false-positive (FP) prediction is the same as the cost for false negative (FN) prediction, values of accuracy and balanced accuracy are almost equal. Also, the difference between true positive rate (TPR) and true negative rate (TNR) as well as the difference between negative predictive value (NPV) and positive predictive value (PPV) is small for supervised methods. The model which performed the worst is, as expected, the semi-supervised AE+GMM model, as for the learning of this model only benign samples were used. The worst performing model among supervised models is the logistic regression (LR) model. The possible justification is that this is the only model which produces linear decision boundary among the selected methods, and this might be inappropriate for the given task. However, the evaluation of models on the same dataset as on which it was trained might be misleading. Hence the results obtained by evaluation of the models on the new set, the test set, will be more informative.

**Table 2 – Evaluation results on test data for all models**

| test | ACC % | BACC % | TPR % | TNR % | FAR % | NPV % | PPV % | $\kappa$ % |
|---|---|---|---|---|---|---|---|---|
| DT | 82.60 | 84.26 | 72.27 | 96.25 | 3.75 | 72.42 | 96.22 | 65.88 |
| RF | 80.66 | 82.67 | 68.17 | **97.18** | **2.82** | 69.79 | **96.96** | 62.37 |
| SVM | 80.65 | 82.30 | 70.38 | 94.22 | 5.78 | 70.65 | 94.15 | 62.08 |
| LR | 75.58 | 77.65 | 62.72 | 92.58 | 7.42 | 65.27 | 91.78 | 52.62 |
| FNN | 80.82 | 82.75 | 68.78 | 96.72 | 3.28 | 70.10 | 96.51 | 62.61 |
| AE+GMM | **89.71** | **89.28** | **92.38** | 86.19 | 13.81 | **89.54** | 89.84 | **78.93** |

Table 2 shows the evaluation results based on the testing dataset for all investigated models. As already mentioned, the testing set consists not only of the attack types which were present in the training set but also includes novel attack types.

Contrary to the previous table of results, there is no method which performed the best in all measures. However, by looking at the most important measure of the binary classification task, namely the accuracy, the semi-supervised approach performed much better than all of the supervised techniques. Even though the proposed semi-supervised model has the highest false alarm rate, i.e., many of the observations classified as attacks are benign, it has also the highest detection rate (true positive rate), i.e., it was the most successful in detecting the positive samples. The difference between TPR and TNR as well as the difference between NPV and PPV is high for all the supervised methods. This means that the supervised method had much higher numbers of false negatives than false positives when predicting using the testing dataset. The possible justification is that the supervised methods were not able to predict novel types of attacks correctly, as such types were not used for the training. The mentioned differences are not that high for the semi-supervised approach, i.e., the value of the FP and FN observations is more alike than in the case of supervised models.

Table 3 shows the percentages of correctly classified test samples with respect to the attack type for all methods. The second and third column show the frequencies of the samples in the test and train set, respectively. We can see that the semi-supervised model (denoted as AE) performed the best in almost all attacks, while some of the attack types were 100% correctly classified by multipleclassifiers.

**Table 3 – Percentage of correctly detected attacks (TPR) by each method**

| label | test | train | DT | RF | SVM | LR | FNN | AE+GMM |
|---|---|---|---|---|---|---|---|---|
| apache2 | 737 | 0 | 27.68 | 62.28 | 69.34 | 99.05 | 79.38 | **99.73** |
| back | 359 | 956 | 98.33 | **100.00** | 60.72 | 13.09 | 81.89 | 27.02 |
| buffer_overflow | 20 | 30 | 10.00 | 15.00 | 30.00 | 50.00 | 65.00 | **100.00** |
| ftp_write | 3 | 8 | **66.67** | 33.33 | 33.33 | 33.33 | **66.67** | **66.67** |
| guess_passwd | 1231 | 53 | 32.41 | 0.00 | 0.16 | 0.24 | 3.17 | **97.32** |
| httptunnel | 133 | 0 | 12.78 | 15.04 | 14.29 | 1.50 | 65.41 | **99.25** |
| imap | 1 | 11 | **100.00** | 0.00 | 0.00 | **100.00** | 0.00 | **100.00** |
| ipsweep | 141 | 3599 | **100.00** | **100.00** | **100.00** | 98.58 | **100.00** | 98.58 |
| land | 7 | 18 | 85.71 | 71.43 | 57.14 | 57.14 | **100.00** | **100.00** |
| loadmodule | 2 | 9 | 0.00 | 0.00 | 50.00 | **100.00** | **100.00** | **100.00** |
| mailbomb | 293 | 0 | 0.00 | 0.00 | 30.03 | 0.00 | 0.00 | **60.41** |
| mscan | 996 | 0 | 90.96 | 75.40 | 86.04 | 51.91 | 76.31 | **100.00** |
| multihop | 18 | 7 | 27.78 | 11.11 | 44.44 | 16.67 | 38.89 | **88.89** |
| named | 17 | 0 | 23.53 | 29.41 | 17.65 | 11.76 | 23.53 | **100.00** |
| neptune | 4657 | 41214 | **100.00** | 99.98 | 99.89 | 99.14 | 99.94 | **100.00** |
| nmap | 73 | 1493 | **100.00** | **100.00** | **100.00** | **100.00** | **100.00** | **100.00** |
| perl | 2 | 3 | 0.00 | 50.00 | 50.00 | 0.00 | **100.00** | **100.00** |
| phf | 2 | 4 | 0.00 | 50.00 | 50.00 | 0.00 | 50.00 | **100.00** |
| pod | 41 | 201 | **100.00** | 95.12 | 90.24 | 95.12 | **100.00** | **100.00** |
| portsweep | 157 | 2931 | 99.36 | **100.00** | **100.00** | 99.36 | **100.00** | **100.00** |
| processtable | 685 | 0 | 61.90 | 22.63 | 48.32 | 0.29 | 13.72 | **100.00** |
| ps | 15 | 0 | 40.00 | 20.00 | 26.67 | 6.67 | 46.67 | **93.33** |
| rootkit | 13 | 10 | 0.00 | 0.00 | 23.08 | 15.38 | 46.15 | **84.62** |
| saint | 319 | 0 | 99.37 | 98.75 | 97.49 | 95.30 | 97.49 | **100.00** |
| satan | 735 | 3633 | 99.59 | **100.00** | 83.67 | 93.33 | 83.27 | **100.00** |
| sendmail | 14 | 0 | 0.00 | 0.00 | 0.00 | 7.14 | 7.14 | **100.00** |
| smurf | 665 | 2646 | **100.00** | **100.00** | 99.25 | **100.00** | **100.00** | **100.00** |
| snmpgetattack | 178 | 0 | 0.00 | 0.00 | 0.00 | 0.56 | 0.00 | **14.04** |
| snmpguess | 331 | 0 | 0.00 | 0.00 | 0.00 | 0.60 | 0.30 | **3.32** |
| sqlattack | 2 | 0 | **100.00** | **100.00** | **100.00** | 0.00 | **100.00** | **100.00** |
| teardrop | 12 | 892 | **100.00** | **100.00** | 83.33 | **100.00** | **100.00** | 91.67 |
| udpstorm | 2 | 0 | **100.00** | **100.00** | 50.00 | 0.00 | 50.00 | **100.00** |
| warezmaster | 944 | 20 | 15.57 | 19.28 | 32.52 | 2.44 | 25.32 | **91.63** |
| worm | 2 | 0.00 | 0 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| xlock | 9 | 0.00 | 0 | 0.00 | 0.00 | 0.00 | 0.00 | **88.89** |
| xsnoop | 4 | 0.00 | 0 | 25.00 | 25.00 | 0.00 | 0.00 | **100.00** |
| xterm | 13 | 0.00 | 0 | 23.08 | 53.85 | 30.77 | 53.85 | **100.00** |

To get a better idea of how good the performance is we compare them to selected publications. Firstly, we have a look at the results from the original publication, where the new subset labelled (NSL)-KDD99 dataset was created [16] to overcome the problems of the original dataset. Afterwards we compare the obtained performance with a recent publication [23], where the authors proposed a transfer learning approach for intrusion detection on the computer network and compared the performance of the proposed model to the other complex deep learning models reported in other publications.

In [16] the authors compared 7 supervised learning techniques. All the methods were trained on only 20% of the KDDTrain+ dataset. The only measure reported is accuracy. Table 4 shows the results. The hybrid technique consisting of Naïve Bayes and decision tree (NBTree) performed the best among the selected methods, and only tree-based algorithms have accuracy on the test set above 80%. Our results are either similar or better in terms of accuracy.

**Table 4 – Results on the KDDTest+ set from the original publication [16]**

| Method | ACC % |
|---|---|
| J48 | 81.05 |
| NB | 76.56 |
| NBTree | 82.02 |
| RF | 80.67 |
| DT | 81.59 |
| MLP | 77.41 |
| SVM | 69.52 |

In [23], the authors propose a supervised transfer learning-based convolutional neural network model (TL-ConvNet) for intrusion detection. The model consists of two convolutional neural networks (CNN). The first CNN,

named base CNN, is trained on the UNSW-NB15 dataset proposed in [11]. The knowledge learned is then transferred and used for the training of the second CNN, which is trained on the KDDTrain+ set, while base CNN is fixed. The whole model is afterwards evaluated on the KDDTest+ set. Accuracy, detection rate and false alarm rate are reported in the publication. Table 5 shows the evaluation measures obtained from the proposed model in percentages. For more details about the other models, the reader is referred to the original publication.

**Table 5 – Results on den KDD99 dataset using transfer learning [23] compared to the AE+GMM model**

| Method | ACC % | TPR % | FAR % |
|---|---|---|---|
| RNN | 83.28 | 72.95 | **0.03** |
| Char-IDS | 85.07 | 81.12 | 9.71 |
| ReNet50 | 79.14 | 69.41 | 0.08 |
| GoogLeNet | 77.04 | 65.64 | 0.08 |
| TL-ConvNet | 87.30 | **93.86** | 21.38 |
| **AE+GMM** | **89.71** | 86.19 | 13.81 |

We can see that our proposed semi-supervised approach AE+GMM (see also Table 2) performed better in terms of accuracy and much better in terms of false alarm rate, than the transfer learning-based approach. The lowest false alarm rate among the compared models is obtained by the RNN model. However neither the detection rate nor accuracy is good. Even though the TL-ConvNet shows a slightly higher detection rate than the proposed semi-supervised approach, it has a much higher false alarm rate. Another advantage of our AE-based model is that it does not require attack samples, which are in many real-world applications either impossible or very expensive to obtain.

## 2.3 Analytical Toolbox

### 2.3.1 Introduction

In this section, we propose a software architecture for the Analytical Toolbox to perform anomaly detection on network traffic using machine learning. It includes components for the complete machine learning lifecycle, i.e.: data pre-processing, model learning, model evaluation, model deployment, and model maintenance.

In data science, such software architectures are typically referred to as machine learning pipelines. These machine learning pipelines aim at automating the execution of machine learning workflows. Solutions are available as cloud-based services (AWS[1], Azure[2], Google Cloud[3], etc.) as well as on-premise services. On-premise solutions range from very basic pipelines, like scikit-learn[4] pipelines, to full-fledged high-performance deep learning pipelines for cluster systems like KubeFlow[5] or TensorFlow Extended[6]. Running these systems on clusters is usually made possible by an orchestration technology like Kubernetes[7].

Since installation and operating of such full-fledged systems is in general not easy, especially for small businesses, the software architecture proposed in this section focuses on individual components that are easy to manage and together also represent a machine learning workflow.

Because data in network traffic can typically be characterized by high volume and high speed, the architecture proposed takes this aspect into account by using components that allow for online anomaly detection on data streams. This includes deploying pre-trained models on data streams as well as performing online learning of models, where models are incrementally updated [7]. Further, a focus is put on how the models can be easily transferred to other settings, e.g., from one shop floor to another in a similar domain.

### 2.3.2 Architecture

Figure 4 shows the building blocks of the proposed architecture. It consists of a data streaming platform, a persistent data storage, and a machine learning model server. Learning of new models or maintaining of existing models is done either in the batch or in the online learning environment. Finally, approved models may be deployed into the model serving environment. In the following only tools are considered for which on premises deployment is available. Equivalent solutions exist for cloud-based deployments as well.

[1] https://aws.amazon.com/machine-learning/

[2] https://docs.microsoft.com/en-us/azure/machine-learning/

[3] https://cloud.google.com/ai-platform

[4] https://scikit-learn.org/

[5] https://www.kubeflow.org/

[6] https://www.tensorflow.org/tfx

[7] https://kubernetes.io/

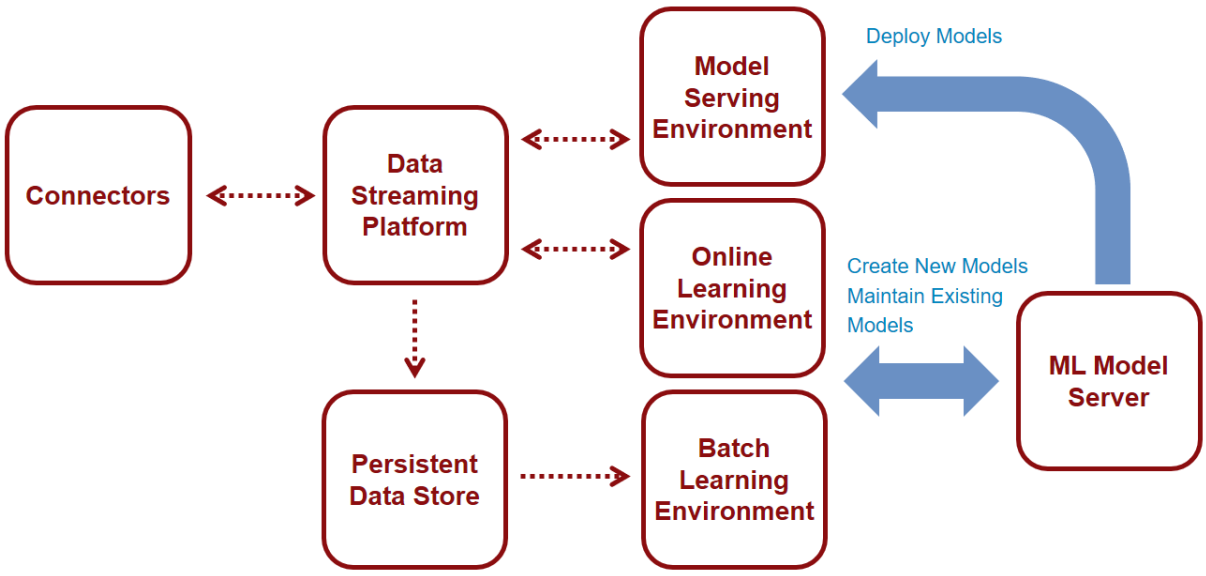



**Figure 4 –Analytical Toolbox Architecture for Anomaly Detection**

### *Data Streaming Platform*

The data streaming platform serves as the central data hub. It collects data from various sources using connectors and makes the data available as data streams to other consumers. Active models in the model execution environment consume this data and also make the result of the model available to other applications by publishing them into separate data streams. The most prominent data-steaming platform is Apache Kafka[8], but there are other alternatives, like Apache ActiveMQ[9], RapidMQ[10] , or Apache Samza[11].

### *Persistent Data Store*

To be able to perform learning on batch datasets, some kind of persistent data store for the training samples is needed. This can range from simple file-based storage or structured relational data storage systems to large distributed NoSQL systems that are usually subsumed under the term big data, like Elastic[12], HBase[13] or Cassandra[14]. Having a persistent data storage is especially important to enable reproducible learning runs.

### *Machine learning Model Server*

A machine learning model server supports the management of learned models, by tracking model configuration parameters and evaluation metrics in a reproducible way and allowing the versioning and comparison of models. As the need for a model management system has been recognized in the last years, multiple commercial and open-source systems are available in the meantime, like MLflow[15] or Kubeflow[16].

[8] https://kafka.apache.org/
[9] https://activemq.apache.org/
[10] https://www.rabbitmq.com/
[11] http://samza.apache.org/
[12] https://www.elastic.co/
[13] https://hbase.apache.org/
[14] http://cassandra.apache.org/
[15] https://mlflow.org/
[16] https://www.kubeflow.org/

#### *Model serving Environment*

There are two possibilities on how to deploy approved models into production [21][20]. Either embed the model natively into the application (e.g., into the data streaming platform or into a numerical control unit) or use a dedicated serving environment, where clients can use the model by issuing calls to the service. Typically, this happens via a RESTful interface. In our architecture, the second possibility, a dedicated serving environment is chosen, since this represents the more general approach to serving models. It is important to note that just serving the pure model is not sufficient. It also requires the deployment of the complete data pre-processing pipeline as well (e.g. applying the same scalers on the features as during learning). Most available machine-learning model servers provide some kind of serving functionality in this style.

Further, since we use data streams as the primary data source, we need a software component that routes the arriving data to the respective machine learning model and that pushes the result back to the data streaming platform. This can be accomplished by building a custom application. For the sake of flexibility, we incorporate Apache NiFi[17] for the task.

#### *Batch Learning Environment*

In the batch learning environment, models are learned or maintained based on samples that are provided from a persistent data storage. It is used for visualizing the data and evaluating the learned models. After the learning has finished, evaluation results together with the learned model can be logged to the model server. Popular machine learning frameworks are scikit-learn, Keras[18], R[19], Matlab[20], etc.

#### *Online Learning Environment*

The online learning environment unifies certain features from the model serving environment as well as from the batch learning environment. Obviously, in this environment, only models can be trained and improved that allow for incremental updates. Supporting machine learning frameworks offer special methods which have to be used for online learning, like the "partial_fit" method in scikit-learn. In addition, the online learning environment must log in regular intervals the status of the online learned model to the machine learning model server.

Compared to the model serving environment, the online learning environment requires substantially more computational resources as learning is much more computationally intensive than just predicting based on an already learned model.

### 2.3.3 Prototypical Implementation

In this section, a prototypical implementation of the software architecture proposed is shown. Apache Kafka is used as a data streaming platform.

The model serving environment is implemented using MLflow and Apache NiFi. While MLflow is used to deploy trained models to a RESTful interface, Apache NiFi is used to orchestrate the dataflow between source data streams, machine-learning models, and target data streams. Figure 5 shows a NiFi dataflow implementing such a model serving environment. The task "Consume from Kafka" consumes data from a source data stream. The task "Transform to JSON-serialized pandas DataFrame" performs data pre-processing and transforms the data into a format that can be consumed by the deployed model. The task "Invoke Prediction Model" performs a call

[17] https://nifi.apache.org/
[18] https://keras.io/
[19] https://www.r-project.org/
[20] https://de.mathworks.com

to the REST API. Finally, the task "Publish to Kafka" publishes the result from the prediction model to a target data stream, from which other applications can make use of the predictions by consuming from it.

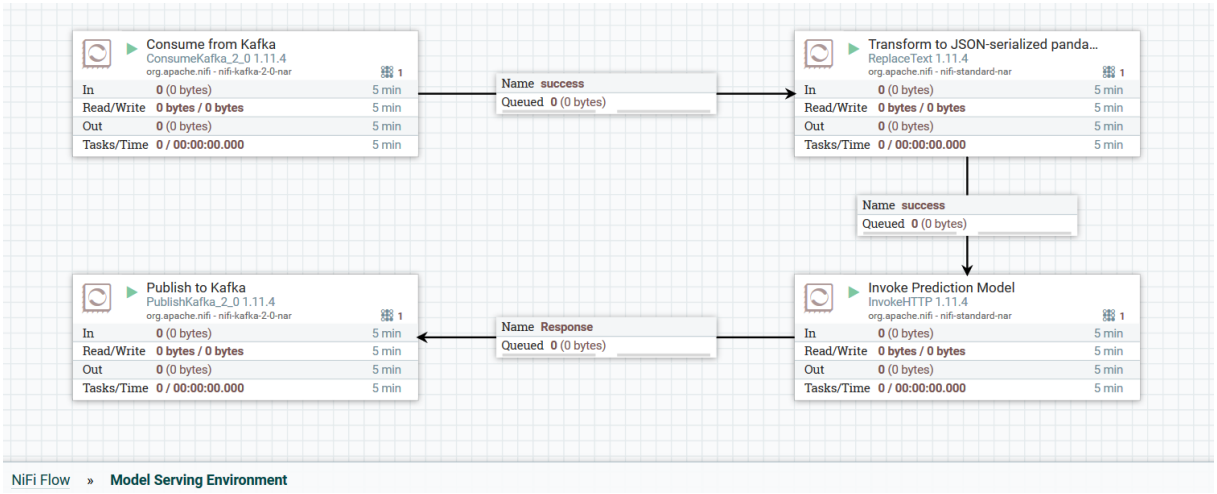



**Figure 5 – Apache NiFi implementation of the model-serving environment**

Also, the online learning environment is implemented using MLflow and Apache NiFi, as depicted in Figure 6. In contrast to the model-serving environment, the task "Merge incoming data to batches" performs a micro batching of the data consumed from the source data streams. In the prototype, the data is aggregated to 1-minute batches, which are then forwarded to the learning task "Incremental Learn Prediction Model". The learning task is implemented in Python and performs an incremental update to an already existing model. It loads the original model from MLflow and after completing the incremental update, it stores the new model together with evaluation metrics in MLflow. Figure 7 shows the sequence of models learned in the online learning environment. Every minute a new model is available.

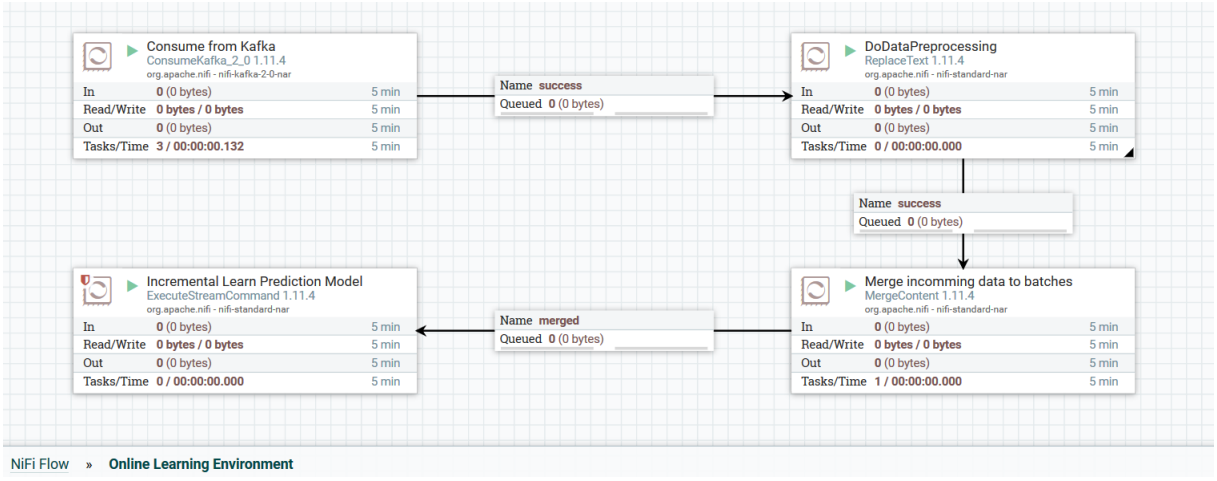



**Figure 6 – Apache NiFi implementation of the online learning environment**

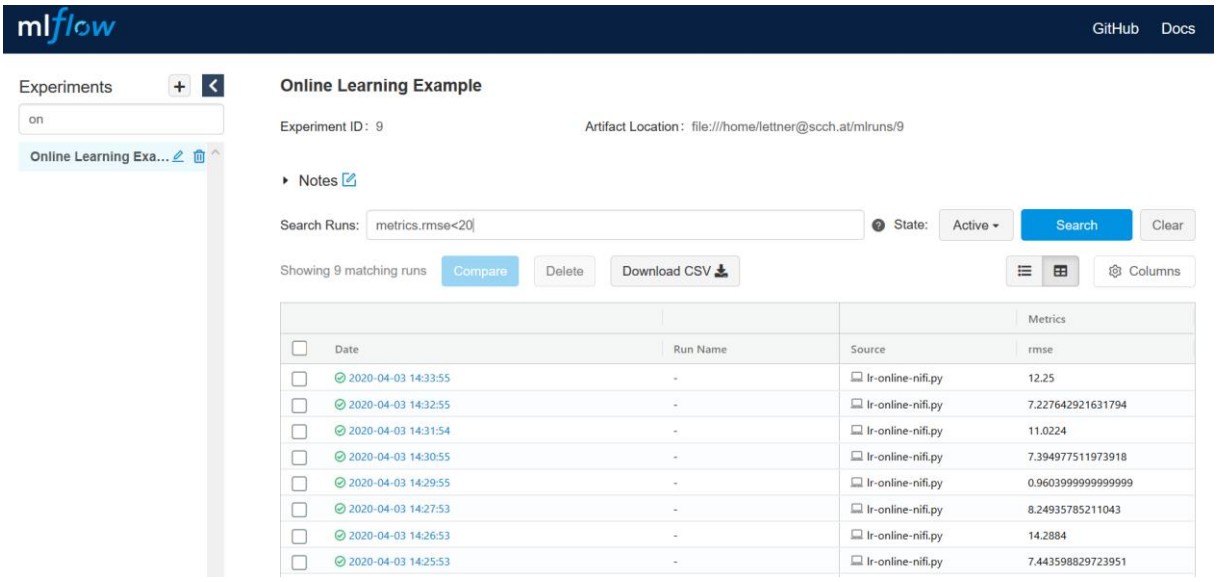



**Figure 7 – MLflow used as a machine learning model server for online learning**

One important function in the batch learning environment is to visualize the available data. We implemented a Shiny[21] application for visualizing the input data allowing for manual (visual) data analysis. Also, the results from learning can be visualized as shown in Figure 8. This shows the reconstruction error for a learned anomaly detector on the KDDTest+ data set as a box plot for all labels (represent different attack types).

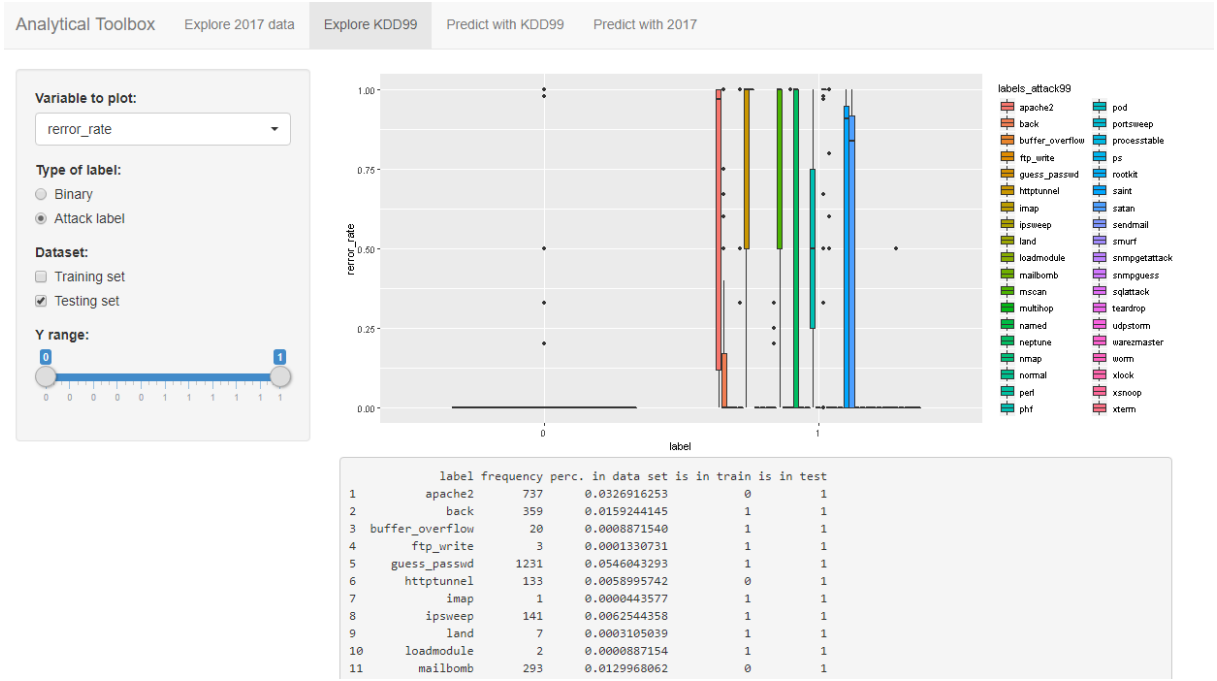



**Figure 8 – Shiny App showing the reconstruction error rate on KDDTest+ data set in a box plot**

[21] https://shiny.rstudio.com/

# 3. Anomaly Detection in Operating Systems

## 3.1 Background

In today's ICT landscape, systems become increasingly interconnected and thus more and more vulnerable to cybersecurity threats. Especially with the emergence of IoT, the attack surface towards malicious attacks is tremendously increased. Traditional approaches to mitigate these threats with intrusion detection systems (IDS), namely signature-based approaches, i.e., black-listing methods, might not be adequate in this new world of interconnected systems of systems, which are often poorly maintained and unmanaged. For example, sophisticated anomaly-based detection mechanisms, in addition to these established systems, can help to mitigate the exploitation of zero-day vulnerabilities, which are hardly detectable by blacklisting-approaches. Furthermore, once the indicators of compromise are widely distributed, attackers can easily circumvent detection of malware. Here, often a simple re-compile with small modifications is enough to put IDS and anti-malware system off track. And the biggest threat, the use of social engineering as an initial intrusion vector with no technical vulnerabilities exploited, gives no indicators that can appropriately be described for a blacklist.

## 3.2 System Log Analysis: Automatic Event Correlation for Incident Detection (ÆCID)

In addition to the anomaly detection approach described in chapter 2 that analyses network traffic data, the system log analysis focuses on processing log data from operating systems and applications. The Automatic Event Correlation for Incident Detection (ÆCID) software system [24], which has been previously developed by AIT, is used as a basis for system log analysis of the Analytical Toolbox. Since ÆCID uses self-learning and white-listing approaches, it is ideal to process logs produced by legacy systems and by appliances with small market shares (like those largely employed in CPS). Furthermore, due to its decentralized architecture with a lightweight component, that can be used on systems with only minimal processing power and memory resources, ÆCID is ideal in an IoT environment.

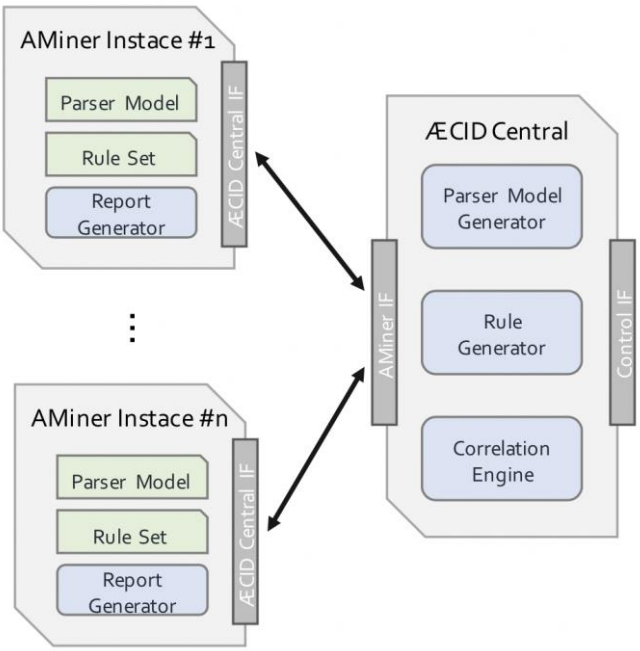



**Figure 9 – ÆCID architecture**

Figure 9 depicts the future system architecture of ÆCID, consisting of the AMiner and ÆCID Central. As mentioned before, ÆCID is designed to allow the deployment in highly distributed environments. Due to its lightweight implementation, an AMiner instance can be installed on any node of the network. The ÆCID Central,

the component responsible of controlling and coordinating all the deployed AMiner instances, can be installed on a host with more resources.

The AMiner operates similarly to a Host IDS sensor and is usually installed on the host and network nodes that needs to be monitored. If available, the AMiner can also be deployed on centralized logging storage which collects the log data generated by the monitored nodes. Each AMiner interprets the log messages generated or collected on a host following a specific model, called the parser model [25]. This model is generated ad-hoc to represent the different events being logged on that node. A custom rule set identifies the events that are considered legitimate in that system. The parser model in combination with the ruleset, describes the expected system behavior of the monitored node. Every log message violating this behavioral model represents an anomaly. In case of an anomaly, the AMiner instance can generate a detailed record of parsed and unparsed lines, alerts and triggered alarms. These reports then can be sent to the ÆCID Central or through additional interfaces to system administrators or other entities. Furthermore, It is also possible to forward all or a subset of events defined by a filter (e.g. to process them in ÆCID Central or a SIEM solution).

ÆCID Central will provide more advanced features compared to the AMiner, which is only used to perform lightweight operations such as parsing incoming log messages and comparing them against a set of pre-defined rules. ÆCID Central can learn the normal system behavior of every monitored system by analyzing the logs received from each AMiner instance and generating a tailored parser model and a specific set of rules in a semi-automatic fashion based on previously received log data. Both then can be used to configure the AMiner instances, in order to detect logged abnormal activities. Additionally, ÆCID Central features a correlation engine that allows analyzing and associating events observed by different AMiner instances, with the purpose of white-listing events generated by complex processes involving diverse network nodes. A Control Interface allows system administrators to communicate with ÆCID Central for its configuration and setting up the individual deployed AMiner instances.

## 3.3 Preliminary Evaluation

### 3.3.1 Setup

This section outlines the results of anomaly detection used with log data received from a larger industrial company. For reasons of confidentiality, the name of the organization is not be named in this document. The setup consists of several components, covering both IT and OT. The IT side includes a firewall, a switch, a Cisco Identity Services Engine, a syslog server, a time recording and a ticketing system. Additionally, a SCADA network with servicing for commissioning and maintenance of automation systems and devices, Master Terminal Units (MTU) and Remote Terminal Units (RTU) provides log data from an OT network.

In order to evaluate the anomaly detection, several attacks have been performed after a training period with only regular system behaviour to generate a viable dataset:

- Attack 1: The attacker gains access to the service program on a device of the network operator. The attacker connects the attacker PCs to the remote-control device.
    - Type A: Configuration change via technician notebook.
    - Type B: Configuration change via the enterprise network with the user not present (determined via time recording, ticketing system)
- Attack 2: The attacker can physically influence sensor values and thus cause local damage.

One of the main components of the anomaly detection system is the parser tree. This parser tree consists of nodes arranged in sequences and branches, with each path through the parser tree representing an event. The nodes can represent both fixed elements, i.e., strings that must appear in the log line at the correct position, and variable elements of a certain data type, for example, an IP address. The anomaly detection features a set of

different detectors with different properties and fields of application. In the simplest case, only the occurrence of values at a single position in the parser tree is considered. The corresponding detector is the `NewMatchPathValueDetector`. Furthermore, it is also possible that these values are considered as combinations. This means that two or more nodes in the parser tree are selected and the common occurrence of values at these nodes is learned. Here, the corresponding detector is the `NewMatchPathValueComboDetector`. Anomalies are also triggered when a log line passes through a path that was specified in the parser tree but never occurred in normal system behavior. In this case, the anomaly is reported by the `NewMatchPathDetector`. Furthermore, the `VariableTypeDetector` learns the distributions of parameter values and continuously adjusts to changes. Finally, the `UnparsedAtomHandler` reports all log lines that do not correspond to the parser tree.

The following variables were used for anomaly detection:

- `NewMatchPathValueDetector`: Source IP in Firewall
- `NewMatchPathValueDetector`: Destination IP in Firewall
- `NewMatchPathValueComboDetector`: Source and Destination IP in Firewall
- `NewMatchPathValueDetector`: Username in Cisco ISE
- `NewMatchPathValueDetector`: List length in Switch
- `NewMatchPathValueDetector`: Source IP in Switch
- `NewMatchPathValueDetector`: Destination IP in Switch
- `NewMatchPathValueComboDetector`: Source and Destination IP in Switch
- `NewMatchPathDetector`: All nodes in the parser tree
- `VariableTypeDetector`: All nodes in the parser tree
- `UnparsedAtomHandler`: Log lines that do not correspond to the parser tree

### 3.3.2 Results

As for **Attack 1, Type A**, the following anomalies were found (Figure 10):

- 12:28: IP events on the switch were detected as unparsed log lines. This is the IP of the attacker laptop. Since these IP events did not occur during normal behavior, they were not included as part of the parser tree. For this reason, the whole line is reported by the `UnparsedAtomHandler`.
- 12:28: A new IP combination was detected on the firewall. This is the IP of the attacker laptop and a broadcast IP.
- 12:30: The IP of the attacker appears again on the firewall. This anomaly could be related to the login.
- 12:31: Four unparsed log lines appeared on the MTU. Three of them are generated during or after the connection is established, the fourth unparsed log line indicates that a password was entered. These log lines are thus linked to the login event.
- 12:35: The MTU reboot is recognized as a large number of unparsed log lines. The `NewPathDetector` reported that new paths of the parser tree have been traversed by SCADA log lines.
- 12:37: The peak here can be explained by the after-effects of the MTU reboot.

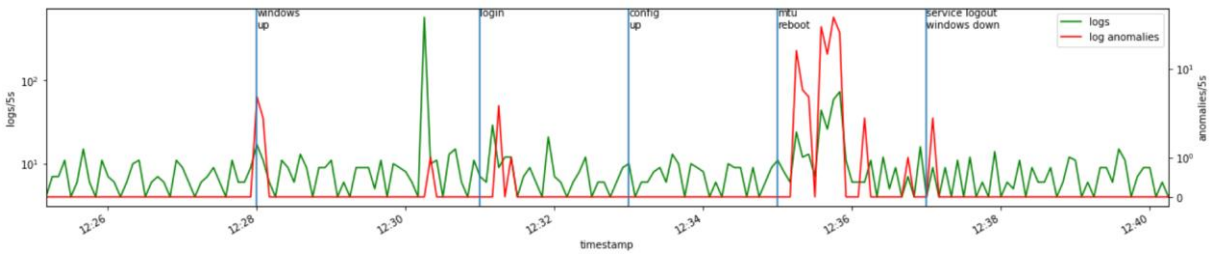



**Figure 10 – Attack 1, Type A: Log anomalies time series**

Conclusion: The activation of the network adapter, the login, and the restart of the MTU were detected. Uploading the configuration and the logout had no noticeable effects which could have led to anomalies.

As for **Attack 1, Type B**, the following anomalies were found (Figure 11):

- 13:12 and 13:14: New services were detected on the firewall. These are identified as anomalies based on the service IDs. Since during normal behavior only certain Service IDs occurred at the affected parts of the parser tree, they were integrated as a fixed part. Thus the new service IDs are reported by the `UnparsedAtomHandler`, because they do not correspond to the parser tree.
- 13:38: The `NewMatchPathDetector` reported that a log line has traversed a previously unused path in the parser tree. This is a path that describes RDP connections. This is a correctly detected anomaly because the connection from the office computer to the service Windows is done via RDP.
- 13:39: The same anomalies of Service IDs have been detected on the firewall a second time. A few seconds later, the combination of two IP addresses was detected as an anomaly by the `NewMatchPathValueComboDetector`, and both IPs were detected as anomalies by the `NewMatchPathValueDetector`. This represents communication with the Service Windows.
- 13:39: A false-positive was detected on the switch. This is because in normal operation almost exclusively more than 1 packet was transmitted (for example "90 packets"). However, only 1 packet is transmitted in this line ("1 packet"), which means that due to the missing "s" in "packets", the line does not correspond to the parser and is reported as unparsed.
- 13:39: Four unparsed log lines were reported, as in Attack 1, Type A, which are due to the login. The unparsed log lines were reported by the reboot with great frequency and new paths were detected on the SCADA systems during the reboot, as well as unparsed log lines and new paths on the firewall.
- 13:45: A new IP address (IP of the Service Windows) was detected on the switch.

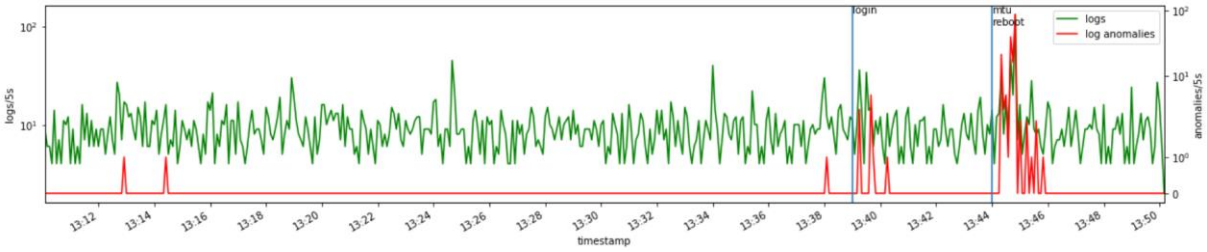



**Figure 11 – Attack 1, Type B: Log anomalies time series**

Conclusion: The attack was successfully detected. This run also provides a good example of a typical false-positive: A very rare event (only 1 packet is transmitted) causes a small change in the syntax of the log lines, and thus an anomaly. However, it should be noted that the change needed in the parser tree to avoid this false-positive in the future is very easy to make and should not be considered problematic.

As for **Attack 2**, the following anomalies were found (Figure 12):

- 14:41: The `VariableTypeDetector` detected an anomaly that is more likely to be false-positive. The anomaly is triggered because the distribution of certain discrete values in log lines changed. This is due to the fact that the changes produced a high number of log lines and thus the distribution change occurred.
- 14:44: The `VariableTypeDetector` detected that the actual sensor value changes. The anomaly contained the information that the previous normal distribution of the value has changed to an unknown, unassignable distribution. This is understandable since at this point in time the sensor value had already changed three times.

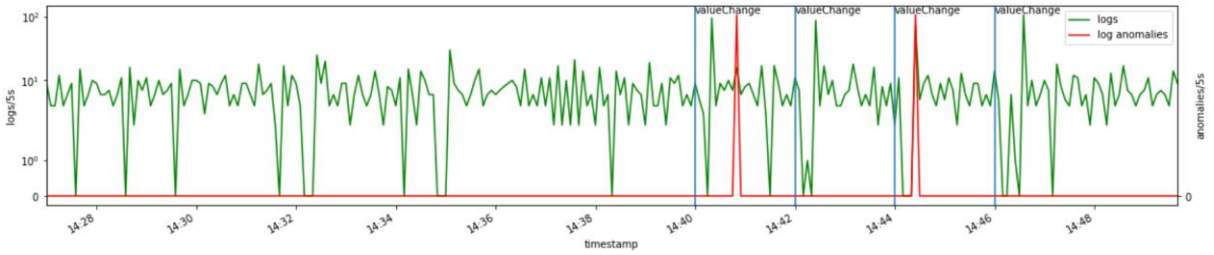



**Figure 12 – Attack 2: Log anomalies time series**

Conclusion: The second anomaly indicates that the attack was successfully detected. The reason the anomaly was not detected immediately after the first change or immediately after each change is because a certain number of values are necessary for a meaningful statistical test. For this reason, detection is delayed until sufficient data is collected.

## 3.4 Analytical Toolbox Integration

In order to integrate log-based anomaly detection into the IoT4CPS Analytical Toolbox, and thus make it available for the demonstrators, some additional steps are necessary. First, a method for storing received log events must be found. Second, a method leveraging the ÆCID components needs to be established, which can be easily integrated without impacting the architecture of the demonstrators. Third, a parser model and a specific set of rules for relevant IoT4CPS components need to be developed.

In order to tackle the first issue, ElasticSearch[22] was chosen to persist log events received from the monitored systems in ÆCID. ElasticSearch is a distributed, RESTful search and analytics engine based on the Apache Lucene library. It is part of the powerful Elastic Stack, an open-source log management solution. Furthermore, ElasticSearch provides a standardized REST-API with ready to use standard libraries for the most common programming languages, including Python, Java, .NET and Perl. This also solves the need for a secure and reliable standardized interface for accessing logs messages from external systems, especially those systems with a high volume of log events. However, for some use cases, using backend-services like MQTT, which already exists in the IoT4CPS demonstrator, might be the more viable option. Especially if the integration effort on the side of the monitored CPS systems must be kept low, and if the number of expected log events is kept within acceptable bounds. Furthermore, Elastic Search allows for asynchronous communication, which is ideal for IoT / CPS environments where a constant Internet connection of all devices cannot be guaranteed. Message queues provide temporary message storage until the recipient retrieves them.

[22] https://www.elastic.co/elasticsearch/

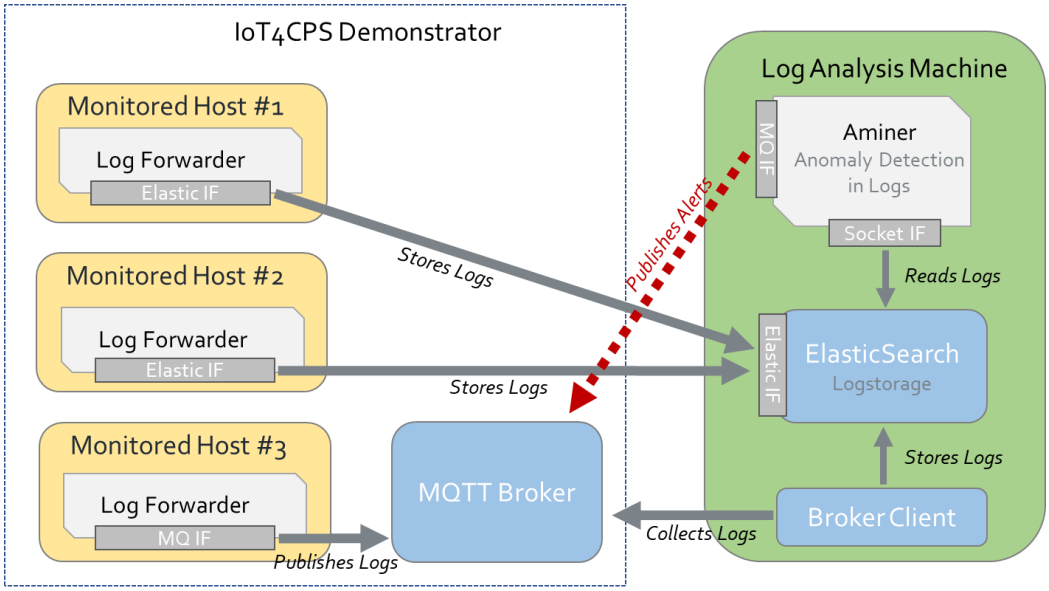



**Figure 13 – ÆCID integration into IoT4CPS**

Therefore, an additional interface for a message queue is integrated. Apache Kafka[23] has been selected for a first proof-of-concept implementation for message queue support. Apache Kafka is an open-source streaming platform and can be used to publish and subscribe to streams of records. It also allows for storing streams of records in a fault-tolerant durable way.

Figure 13 gives an overview of the integration to the Analytical Toolbox. The AMiner serves as a central processing component for detecting anomalies in received log events. ElasticSearch is used for both persisting logs and providing a RESTful interface for storing log events from external systems. An additional broker-client accesses logs distributed through the central IoT4CPS MQTT Broker and stores them at the ElasticSearch instance for further analysis. In the IoT4CPS demonstrators, two possibilities exist to use the log analysis capabilities of ÆCID:

1) **Monitored Host #1 & #2** shows the standard approach, using the ElasticSearch RESTful API to store single log events at the ElasticSearch instance. This can be done by either using the API directly or by configuring a FileBeat[24].
2) **Monitored Host #3** shows the use of a simple log forwarder, which uses MQTT for pushing log files to a MQTT broker. This component will have to be newly developed for IoT4CPS.

Anomalies detected by AMiner are disseminated through the central MQTT broker. This allows for easy integration into monitoring systems like SIEMs and ensures that adequate measures can be taken to respond to security incidents.

[23] https://kafka.apache.org/

[24] https://www.elastic.co/beats/filebeat

## 4. Hardware Anomaly Detection

The interconnectivity between several devices has raised security issues in IoTs, especially in the network layer. The security issues in network layers include information stealing, communication channel jamming, spoofing, denial-of-service, etc. Traditionally, these security issues are addressed at the application layer, protocol layers, or system level. However, these techniques cannot ensure the trustworthiness of each component. For example, in the case of compromised hardware, i.e., a small piece of hardware that performs a security attack when it gets a trigger, known as hardware Trojan (HT), these techniques cannot guarantee the privacy of information. Therefore, it is imperative also to ensure the security of the communication at the hardware level.

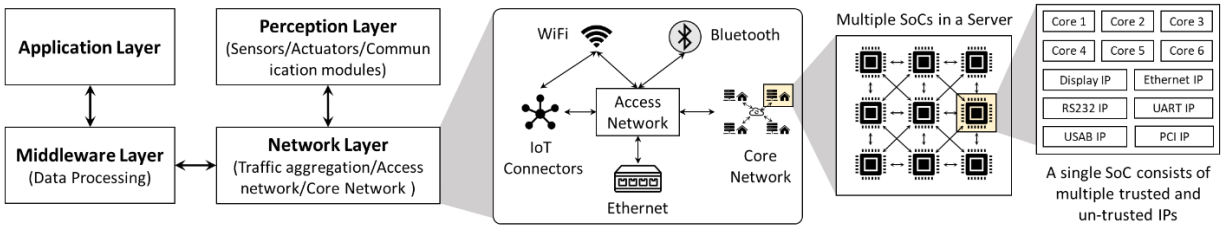



**Figure 14 – Thematic focus of the proposed technique that introduces the formal hardware checks to secure the SoCs**

The main focus of this work is to provide the security of the basic building block of every component in the network layer, i.e., system-on-chip, as shown in Figure 14**Fehler! Verweisquelle konnte nicht gefunden werden.**. SoC consists of several trusted and untrusted 3rd party intellectual properties. State-of-the-art HT detection techniques utilize side-channel parameters, i.e., timing, power, current or electromagnetic signals, based on the golden signatures[25] to detect anomalous behaviour. However, in the case of 3rd party intellectual property based designs, it is nearly impossible to extract the golden signatures because intellectual properties can already be un-trusted. To address this issue, various intellectual property analysis-based approaches have been proposed, but they inherently pose the following limitations:

1) Limited access to the intellectual properties and measurement inaccuracies can compromise the accuracy of the golden signatures.
2) Reverse engineering-based techniques are costly, and the existing sensors-based techniques cannot encompass all the possible input conditions because of the inherent quantization loss of analog-to-digital conversion (ADC).

To address the above-mentioned limitation of coverage of all input combination, different Machine Learning (ML)-based techniques have been proposed in the literature. Those methods train the ML models with communication patterns or power profiles. However, these techniques either possess a large overhead of ML computations or a large overhead of run-time data acquisition, both of which would be infeasible in resource-constrained edge devices, especially under environmental and process variations. Therefore, these issues raise a key research question: how to enable a lightweight ML-based HT detection technique, and consequently, what is the associated run-time data acquisition overhead and the sensitivity to the process variations?

To address the above research question, we propose a novel methodology, called MacLeR, to design an ML-based run-time HT detection technique that exploits the fine-grained power profiling of the microprocessor. Towards this, MacLeR employs the following analysis and methods:

[25] The golden signatures represents the normal or baseline behaviour w.r.t. a particular property of the system.

1) To obtain the fine-grained power profiles, we propose to measure the individual power of each pipeline stage w.r.t. a particular instruction. The reason for choosing this is that the impact of HTs on fine-grained power profiles is relatively better noticeable as compared to the overall power.
2) To reduce the complexity and detection time, we propose an off-chip monitor that collects analog power profiles and converts them into the digital domain. These power profiles are then used first for training an ML model (in our case it is lightweight multi-layer perceptron (MLP)) at design time, and afterward at run time for detecting HTs. The reason for choosing MLP is because it requires fewer computations and is typically faster as compared to other complex ML algorithms.
3) To extract the fine-grained power profiles of a microprocessor during the runtime requires multiple power ports. Therefore, to reduce the number of power ports, we propose a single power-port current acquisition block (SP-CAB) and accordingly measure the current in a time-division multiplexing manner.
4) To study the robustness of MacLeR (i.e., drop in HT detection accuracy), we perform a sensitivity analysis under the process variation by performing the Monte-Carlo simulation using the PV models from TSMC 65nm technology.

## 4.1 Targeted Threat Model

We assume that the 3rd party intellectual property vendors are not trustworthy, and therefore, the specification and source code provided by the vendor may contain HTs. The hardware designers/architects who integrate different 3rd party intellectual properties, along with the in-house intellectual properties (if any), to develop an SoC are considered to be the defenders. Note that among these intellectual properties at least one intellectual property is trusted[26]. This work targets the HTs, which have a direct or indirect impact on the shared power network between the intellectual properties of an SoC. Although the multi-core SoC can have multiple power grids and different voltage islands, the SoCs in battery-operated components for edge devices typically have only one power grid with multiple voltage islands, which is shared between different components. Therefore, in this work, we design a low-power machine-learning-based run-time HT detection methodology for such SoCs that typically have one power grid with multiple voltage islands.

## 4.2 Methodology

Figure 15 shows the complete step-by-step flow of McLeR, which consists of the following key steps:

1) The main goal of MacLeR is to design a low-power ML-based monitor for HT detection, for which a proper training dataset is required. Towards this, during the design phase, first, MacLeR generates the power profiles by measuring the combined power consumption of each component involved in a particular pipeline stage. However, to generate the abnormal power profiles of a microprocessor for MLP training, we use the Trust-Hub HT benchmarks. Note, we use different sets of HT benchmarks for training and testing of MacLeR to avoid any kind of training bias.
2) During the design phase, it uses the generated power profiles to train different variants of MLPs. Then it performs a design space exploration (DSE) w.r.t. the detection accuracy and the associated overhead of the MLP, and it chooses the most appropriate MLP, which is used for run-time HT detection.
3) During the run time, MacLeR measures the power consumption of each component involved in a particular pipeline stage using multiple current mirrors[27]. Then it collects the combined power using a single pMOS transistor. Finally, these power values are collected in a time-division multiplexing manner and used by the trained MLP (which is the best-one obtained from the Step-2 of DSE) to detect HTs.

[26] In this assumption, the trusted intellectual property means that it is designed by an in-house trusted designer.

[27] Current mirror is an analog circuit that copies the current of an active device using a diode-connected CMOS transistor. Typically, these circuits are used to sense the current of an active device.

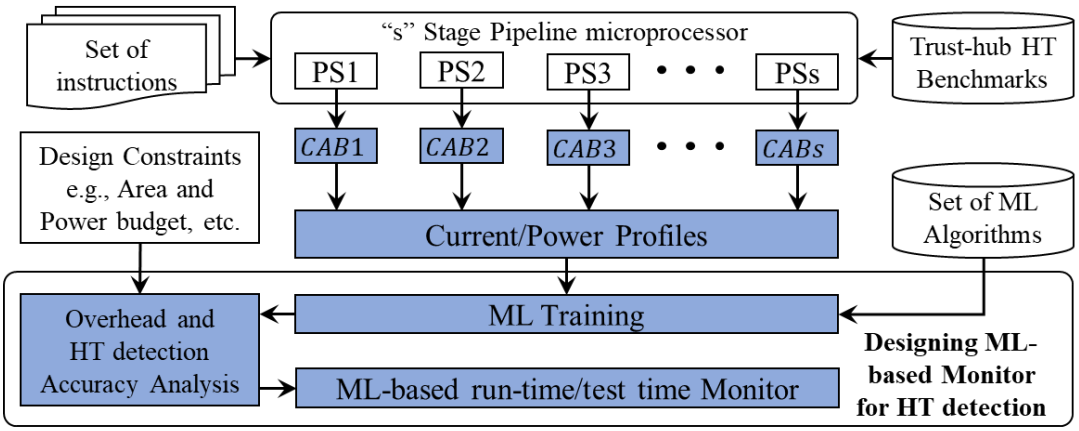



**Figure 15 – MacLeR: ML-based methodology to run-time power monitoring. CABs represent the current acquisition block for s$^{th}$ pipeline stages (PSs).**

## 4.3 Hardware Implementation

During design time, fine-grained power profiles are obtained by measuring the instruction-dependent power of each component associated with a pipeline stage. However, power acquisition during the run-time poses a research challenge about how to acquire the fine-grained power profiles with a minimum area overhead?

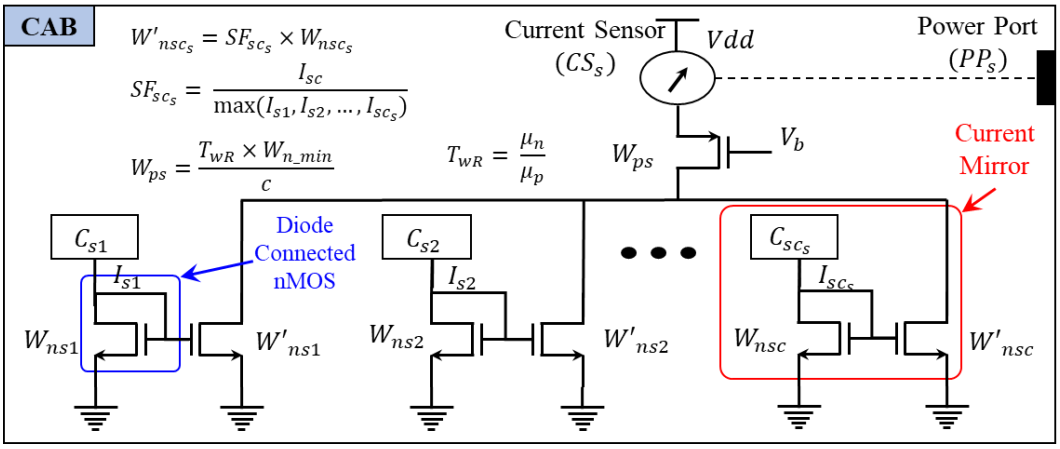



**Figure 16 – The proposed single power-port current acquisition block (CAB).**

For this, we propose to use multiple current mirrors for measuring the current from each component and collect it using a single pMOS transistor-based collector, as shown in Figure 16.

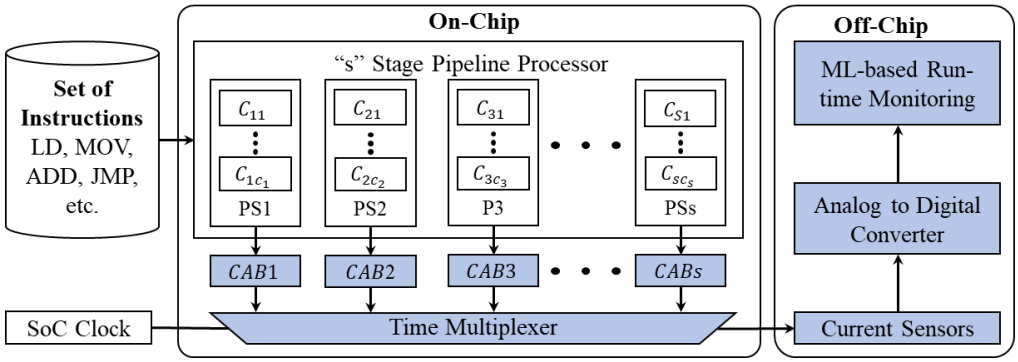



**Figure 17 – Single power-port based hardware implementation of the MacLeR with on-chip and off-chip hardware components.**

The single current sensor-based power-ports are used to measure the current of different modules involved in each pipeline stage. Typically, the number of power-ports required to cover all the modules is equal to the number of components involved in the operations of pipeline stages, e.g., in n-stage pipeline architecture, the number of power-ports is $N_p = C_1 + C_2 + \ldots + C_n$, where $C_1, C_2, \ldots, C_n$ are the number of components involved in pipeline operations of $1^{st}, 2^{nd}, \ldots, n^{th}$ pipeline stage, respectively. Having multiple power ports in an IC is very expansive for the IC packaging. Therefore, to acquire the complete power profile of the microprocessor via a single power-port, we propose to acquire the data using time multiplexing (which measures the current of each SP-CAB after every clock cycle), as shown in Figure 17. Note, MacLeR extracts and uses the fine-grained power profiles of a trusted microprocessor in an SoC. Therefore, it does not require any golden circuits of un-trusted intellectual properties.

After acquiring the power profiles of the microprocessor, MacLeR chooses an appropriate ML algorithm based on the required HT detection accuracy and design constraints. Towards this end, we propose an iterative methodology that first trains the multiple configurations of MLPs using the following steps, as shown in Figure 15.

1) We start by labelling different power-profiles to differentiate the malicious and benign power profiles. These labels are, in turn, used to train and validate the ML models. Note, during the design time, the abnormal power profiles are obtained using the trust-hub HT benchmarks.
2) Next, we categorize these power profiles w.r.t. the functional and behavioural similarity to increase the efficiency of the ML models.
3) After the categorization, we train the multiple ML models and validate them by applying the testing dataset.
4) Finally, MacLeR selects the best MLP model based on the maximum HT detection rate, associated overhead, and the given design constraints.

## 4.4 Experimental Results

We illustrate the practicality, utilization, and effectiveness of our MacLer framework by applying it on a LEON3-based SoC, where at least one of the intellectual properties is trusted, as shown in Figure 18. The main motivation of choosing LEON3 is that it is highly configurable and open-source, and the HT benchmarks that are provided by trusthub.org [27] can easily be integrated into it. The workloads used for LEON3 are 64-bit encrypted multiplication, subtraction, addition, and division. The inputs are encrypted using AES, and results are displayed on the screen using VGA and also transmitted using Ethernet and RS232 intellectual properties. Note, the addition, subtraction, and multiplication are used for training the ML monitor, and the division is used at the inference stage of the ML-monitor to detect the HTs, and therefore, avoiding the biasing in the testing phase.

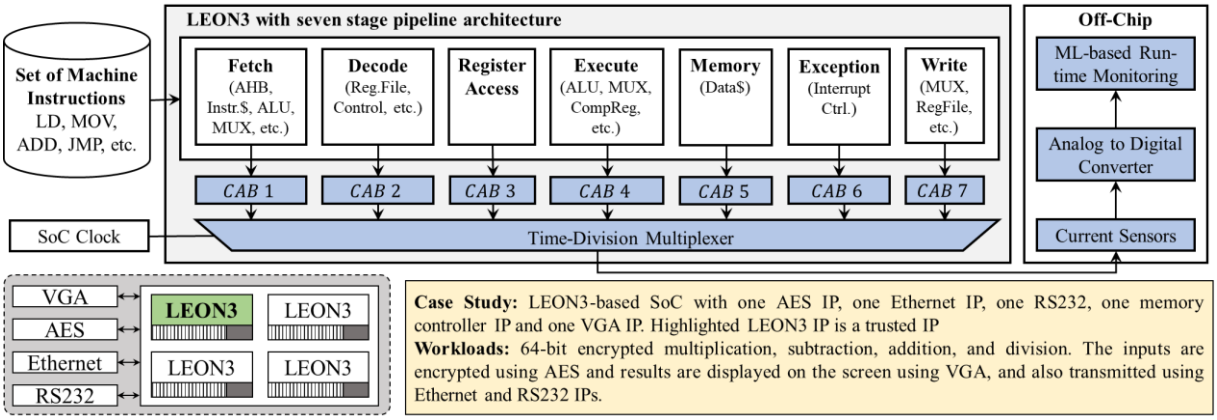



**Figure 18 – Hardware implementation of the MacLeR monitoring framework for LEON3 with a seven-stage pipeline architecture.**

### 4.4.1 LEON3: Power Profiling and Data Acquisition

For obtaining the power profile, we synthesized the LEON3 processor using Cadence Genus (Encounter) tool with the TSMC 65nm library. The power of each module involved in the pipeline stage is calculated separately for each instruction. For example, Figure 19 shows the power consumption in each pipeline stage for a particular instruction, which is extracted by executing one instruction at a time.

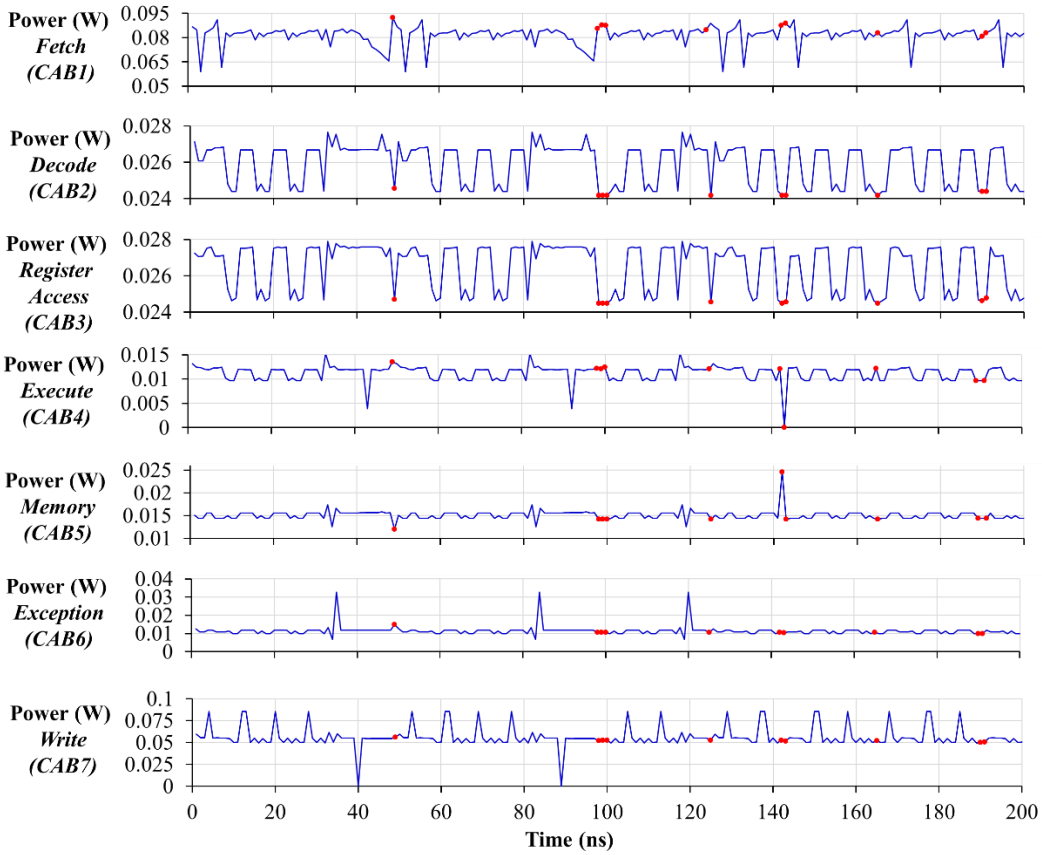


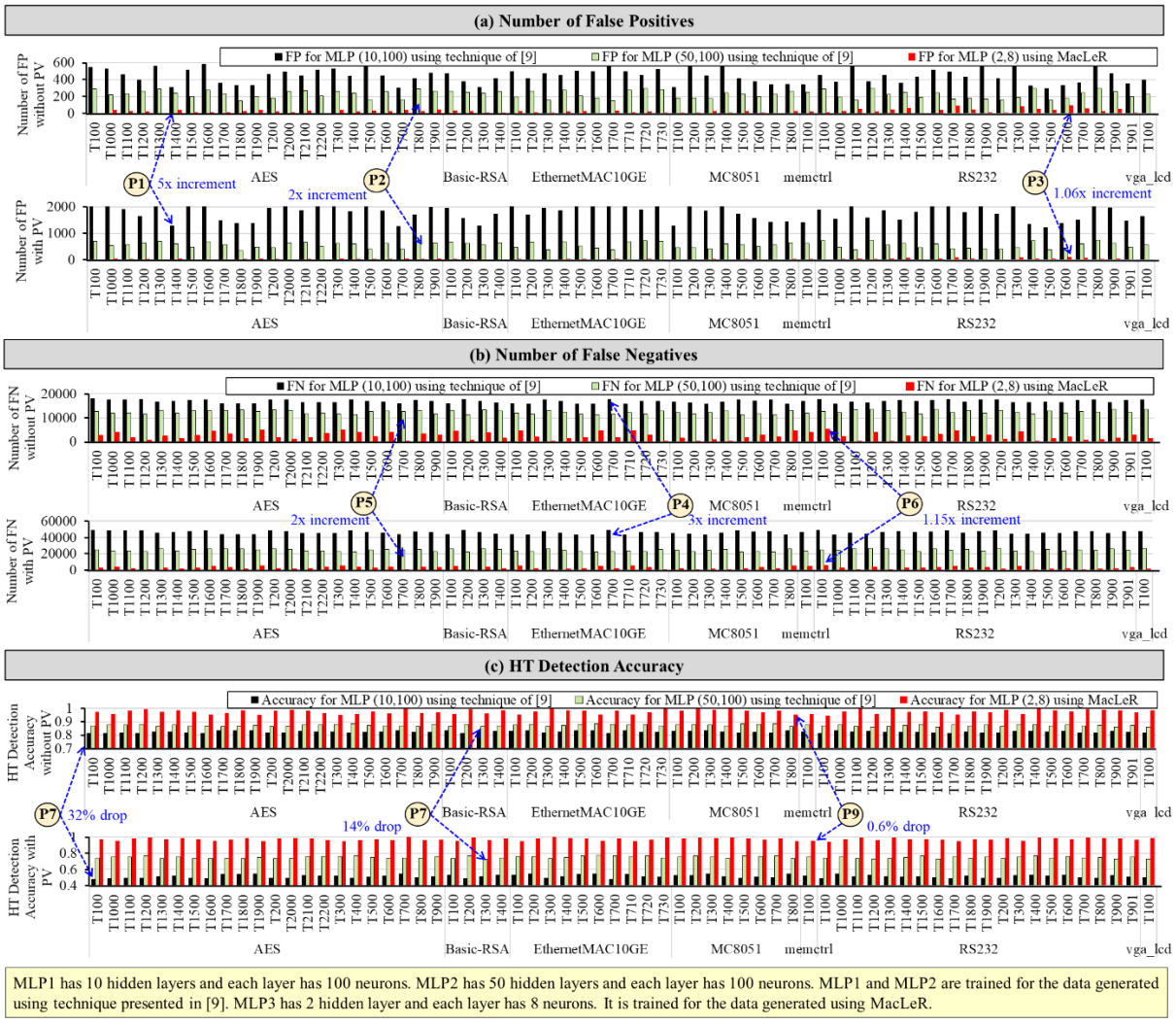

**Figure 19 – Power consumption of different pipeline stages of the LEON3 microprocessor while executing multiple instructions. The Red dots in the figure represent the activations of MC8051-T200 in the SoC.**

The power consumption of a microprocessor is dependent upon the number of modules involved in the execution of instructions during a particular pipeline stage, e.g., LEON3 has seven different power profiles, i.e., one for each pipeline stage. To model the power behaviour, first, we identify all the modules involved in the operation of a particular pipeline stage. Figure 18 shows that in LEON3, the fetch stage requires instruction cache (I-Cache), AHB bus, adder, and multiplexers (MUX). Decode, register access, execute, memory, and exception stages require register file, ALU, data cache (D-cache), and interrupt controller. In LEON3, the fetch stage consists of 4 components, which is the maximum number of components involved as compared to any other pipeline stage. In our experimental setup, AES-T100, AES-T800, vgalcd-T100, RS232-T1000, memctrl-T100, and ethernetMAC10GE-T700 benchmarks are used to generate the data for training, while the data generated using all the HT Trojan benchmarks for RS232, MC8051, AES, Basic RSA, VGA-LCD, memory controller and Ethernet intellectual properties are used for evaluation.

Figure 19 shows the power profiles of LEON3 in the presence of MC8051-T200 in SoC, and the red dots show the triggering of MC8051-T200 . The power value at each time unit, along with instruction and its category, is extracted to generate the required power profile. For example, at 1ns the power profile consists of the power of each pipeline stage is ([0.086649W, 0.027123W, 0.027238W, 0.013182W, 0.015111W, 0.012424W, 0.059794W], LD and Cat1). These profiles are then used to train the ML model in the next phase.

### 4.4.2 LEON3: Run-time Monitor for HT Detection

After extracting the power profile in the previous phase, we analyse and transform it in such a way that it can be utilized for training and validation. This involves proper labelling of the extracted power behaviour to differentiate between malicious and benign behaviours, and pre-processing to reduce the redundant dataset. After the transformation, we trained different configurations of neural networks (multi-layer perceptrons (MLP) with one and two hidden layers. Afterward, we analysed the trade-off between accuracy and computational cost. The experimental analysis shows that in the case HT benchmarks for MC8051, the trained MLP with two hidden layers and eight neurons in one layer provides approximately 98% HT detection accuracy, and with a very small number of false positives and false negatives. However, for other HT benchmarks, MacLeR still provides approximately 90% HT detection accuracy.




MLP1 has 10 hidden layers and each layer has 100 neurons. MLP2 has 50 hidden layers and each layer has 100 neurons. MLP1 and MLP2 are trained for the data generated using technique presented in [9]. MLP3 has 2 hidden layer and each layer has 8 neurons. It is trained for the data generated using MacLeR.

**Figure 20 – False positive, False-negative and HT detection accuracy of the MacLeR and the state-of-the-art run-time ML-based HT detection technique [10], in the presence of different HT benchmarks with and without considering the process variations (PV). Note, these analyses are based on the 100,000 classification per HT, where the total number of activations is 1000 out of 100,000. Note, in these experiments, all the benchmarks related to MC8051 are implemented in the LEON3 microprocessor, and all other HT benchmarks are also configured for the LEON3 microprocessor.**

### 4.4.3 Sensitivity to the Process Variations

The side-channel parameters-based (like power consumption) HT detection techniques are vulnerable to process variations (PV). Therefore, we also analyze the impact of PV on the MacLeR by performing Monte-Carlo

simulation using given PV models from TSMC 65nm technology. Key results of this sensitivity analysis (see Figure 20, Figure 21, and Figure 22) are:

1) In the microprocessor, the power variations due to HT are significantly higher as compared to the PV-induced power variations.
2) The average drop in the HT detection accuracy of MacLeR is less than 1% (96.256% to 95.85%), which is approximately 10x less than that for the state-of-the-art [10] when subjected to PV.

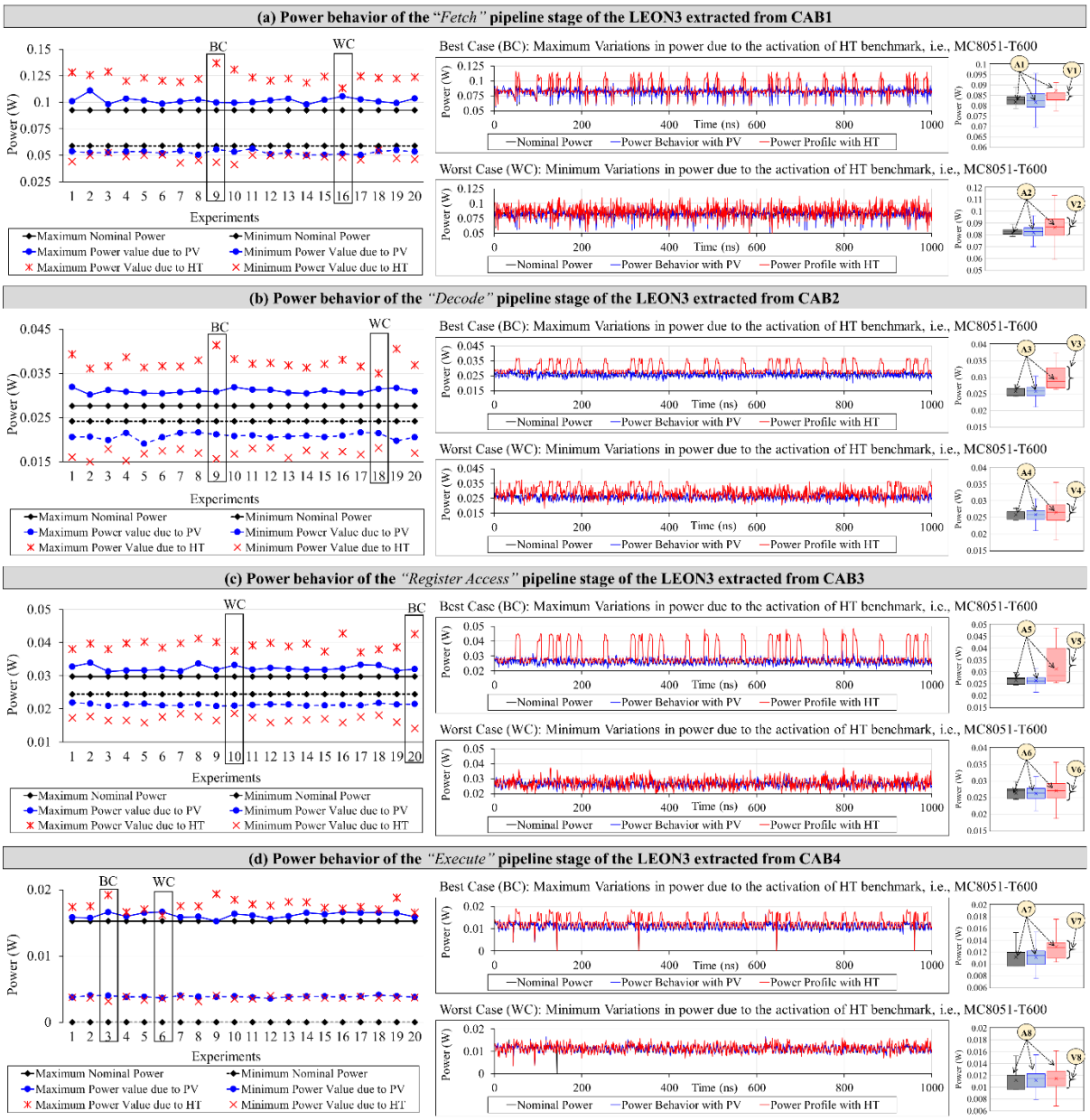



**Figure 21 – The impact of process variations and implemented hardware Trojan benchmark, i.e., MC8051-T600, on the power behavior of Fetch, Decode, Register Access, and Execute pipeline stages of LEON3 microprocessors while executing the multiple instructions. The results presented in black, blue, and red colors represent the nominal power behavior, variations in power behavior due to PV, and variations in the power behavior due to HT, respectively. Note, in these analyses, Best-Case (BC) defines the scenarios in which HT is easily detectable, and Worst-Case (WC) defines the scenarios in which HT is hard to detect.**

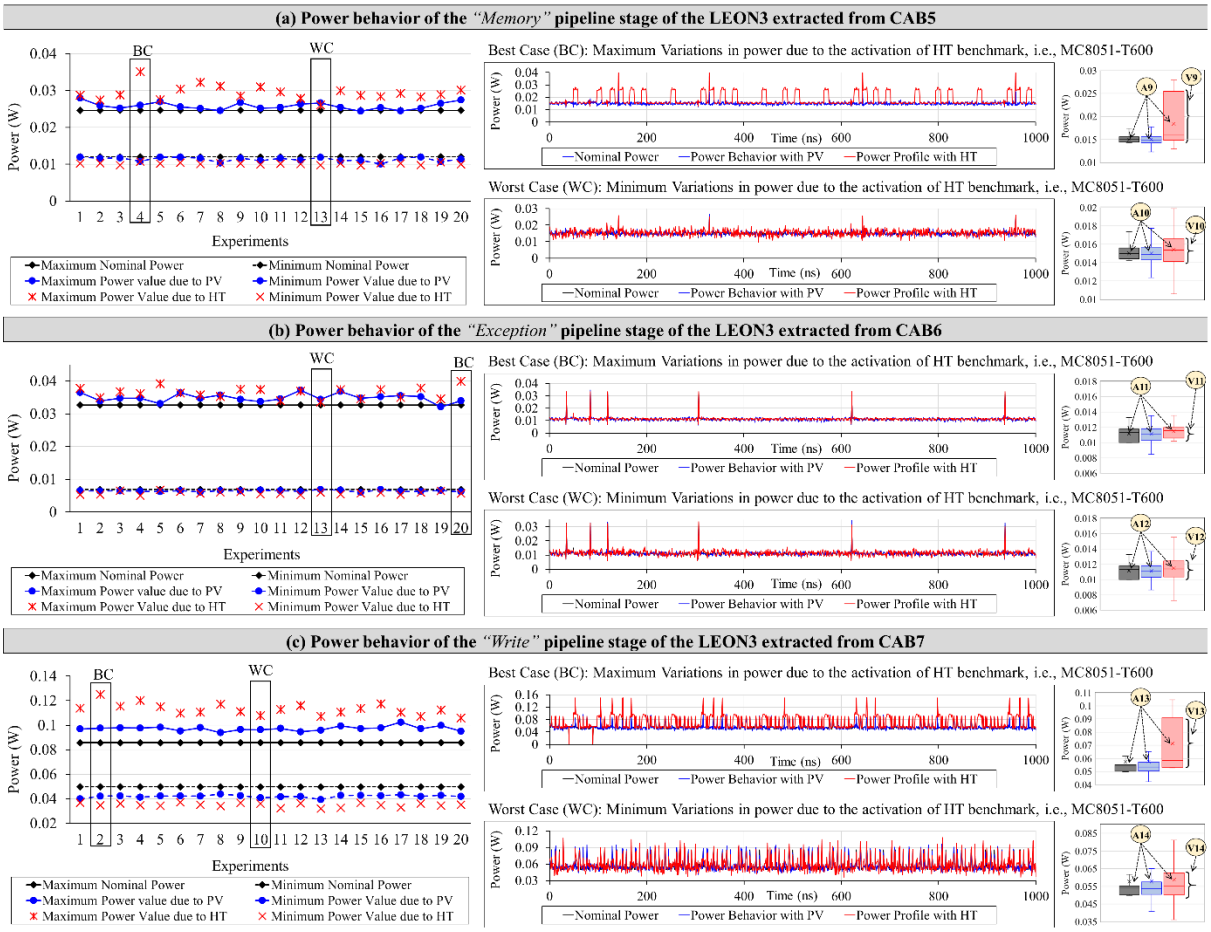



**Figure 22 – The impact of process variations and implemented hardware Trojan benchmark, i.e., MC8051-T600, on the power behavior of Memory, Exception, and Write pipeline stages of LEON3 microprocessors while executing the multiple instructions. The results presented in black, blue, and red colors represent the nominal power behavior, variations in power behavior due to PV, and variations in the power behavior due to HT, respectively.**

### 4.4.4 On-Chip Overhead of our New Hardware Components of MacLeR

To analyze the overhead of the proposed MacLeR methodology, we synthesized the RTL of the complete LEON3-based SoC using a 65nm technology in Cadence Genus. The overall area overhead for SP-CAB, time multiplexing is approximately 7.01$\mu m^2$, which is less than 0.15% of the total area, thus negligible. Similarly, the power overhead is approximately 15 $\mu$W, which is less than 1% of the total power, again negligible.

# 5. Overall Summary

To summarize this phase of the project, we have finalized on anomaly detection within IoT4CPS. Software as the main outcome has been developed and build the basis for the future application of anomaly detection on the use cases and for real-world scenarios. This includes:

- Continuous development of the ÆCID framework within the Analytical Toolbox. In particular, the integration of a message queue into ÆCID and accessing log events from ElasticSearch have been implemented and successfully tested.
- Machine learning methods for anomaly detection on network traffic for detecting attacks and intrusions where developed and tested on the publicly available datasets. Further, a prototype of the Analytical Toolbox, which allows us to perform online learning, has been implemented.
- A machine learning-based technique that uses fine-grained power profile for run-time hardware Trojan detection.

# 6. References

[1] Abdulhammed, Razan, “Features Dimensionality Reduction Approaches for Machine Learning Based Network Intrusion Detection.” *Electronics*, vol. 8, no. 3, Mar. 2019, article nr. 322.

[2] Adee, Sally. "The hunt for the kill switch."iEEE SpEctrum 45.5 (2008): 34-39.

[3] Almseidin, Mohammad, “Evaluation of Machine Learning Algorithms for Intrusion Detection System.” Proc. of the IEEE 15th International Symposium on Intelligent Systems and Informatics (SISY), 2017, pp. 277–282.

[4] Chen, Wun-Hwa, “Application of SVM and ANN for Intrusion Detection.” Computers & Operations Research, vol. 32, no. 10, Oct. 2005, pp. 2617–2634.

[5] Collins, Michael. Network Security through Data Analysis: From Data to Action. Second edition, O’Reilly Media, 2017.

[6] Frazier, P.: A tutorial on bayesian optimization, 2018

[7] Grubinger, T., Chasparis G., Natschläger T.: Online transfer learning for climate control in residential buildings. Proceedings of the 5th Annual European Control Conference (ECC'16), pages 1183-1188, EUCA, June, 2016.

[8] Klambauer, G. : Self-Normalizing Neural Networks. Neural Information Processing Systems, 2017

[9] Laskov, Pavel, “Learning Intrusion Detection: Supervised or Unsupervised?” Proc. of the Image Analysis and Processing (ICIAP), 2005, pp. 50–57.

[10] Lodhi F. K. et al., “Power profiling of microcontroller’s instruction set for runtime hardware trojans detection without golden circuit models,” in IEEEDATE, 2017, pp. 294–297.

[11] Moustafa, Nour, and Jill Slay. "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)." 2015 military communications and information systems conference (MilCIS). IEEE, 2015.

[12] Panigrahi, Ranjit and Samarjeet Borah. “A Detailed Analysis of CICIDS2017 Dataset for Designing Intrusion Detection Systems.” International Journal of Engineering & Technology, vol. 7, no. 3.24, 2018, pp. 479–482.

[13] Sabhnani, Maheshkumar, and Gursel Serpen. “Why Machine Learning Algorithms Fail in Misuse Detection on KDD Intrusion Detection Data Set.” Intelligent Data Analysis, vol. 8, no. 4, Oct. 2004, pp. 403–415.

[14] Sharafaldin, Iman, “Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization:” Proc. of the 4th International Conference on Information Systems Security and Privacy (SCITEPRESS), 2018, pp. 108–116.

[15] Švihrová Radoslava: "Machine Learning Techniques for Intrusion Detection in Network Security". Master thesis submitted at Institute of Applied Statistics, Johannes Kepler University Linz, 2020

[16] Tavallaee, M., “A Detailed Analysis of the KDD CUP 99 Data Set.” Proc. of the IEEE Symposium on Computational Intelligence for Security and Defense Applications, 2009, pp. 1–6.

[17] Tehranipoor M. and H. Salamani H., “trust-HUB,” 2016. [Online]. Available: https://www.trust-hub.org/

[18] Vapnik, V.: Statistical learning theory, 1998

[19] Valiant, Leslie G. "A theory of the learnable." Communications of the ACM 27.11 (1984): 1134-1142.

[20] Villasenor J. and Tehranipoor M., “The hidden dangers of chop-shop electronics: Clever counterfeiters sell old components as new threatening both military and commercial systems,” IEEE Spectrum (cover story), 2013.

[21] Waehner, Kai; Machine Learning and Real-Time Analytics in Apache Kafka Applications; https://www.confluent.io/blog/machine-learning-real-time-analytics-models-in-kafka-applications/ accessed: 26.3.2020

[22] Wang, Yun. Statistical Techniques for Network Security: Modern Statistically Based Intrusion Detection and Protection. Information Science Reference, 2009.

[23] Wu, P.: A transfer learning approach for network intrusion detection, 2019

[24] Wurzenberger, M., Skopik, F., Settanni, G., & Fiedler, R.: AECID: A Self-learning Anomaly Detection Approach based on Light-weight Log Parser Models. In ICISSP (pp. 386-397), 2018

[25] Wurzenberger, M., Landauer, M., Skopik, F., & Kastner, W. AECID-PG: A Tree-Based Log Parser Generator To Enable Log Analysis.

[26] Zhao, Yang, et al. "Memory Trojan Attack on Neural Network Accelerators." 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2019.

[27] A Community Resource for Archiving Wireless Data at Dartmouth. Website crawdad.org/, May 31.