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):	Patrick Traxler
E-mail:	patrick.traxler@scch.at
Author(s):	Patrick Traxler, Arndt Bonitz, Faiq Khalid
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

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

Federal Ministry
Republic of Austria
Transport, Innovation
and Technology

FFG
Forschung wirkt.

Content

Abbreviations

ÆCID	Automatic Event Correlation for Incident Detection
CPS	Cyber-physical system
ICT	Information and communications technology
IDS	Intrusion Detection System
IoT	Internet of Things
HIDS	Host IDS
HT	Hardware Trojans
MQ	Message Queue
MQTT	MQ Telemetry Transport
PSL	Property Specification Language
SoC	System on a Chip
3PIP	3rd Party Intellectual Property

Executive Summary

This deliverable describes the current progress on anomaly detection for IoT4CPS and its intended implementation of the Analytical Toolbox. Results have been achieved by the IoT4CPS partners AIT, SCCH and TU Vienna during this first 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, the integration of a message queue into ÆCID, have been undertaken und successfully tested. ÆCID stands for Automatic Event Correlation for Incident Detection and is an intelligent cyber security 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 anomaly detection on the network communication level. A first achieved insight is that decision tree learning seems to be particular well suited for attack and intrusion detection problems at the network communication level. This work has been carried out on public datasets for intrusion and attack (anomaly) detection. In addition, a Shiny app has been developed to test first data visualization techniques for the Analytical Toolbox. The work of AIT and SCCH is complemented by the TU Vienna which mostly concerns anomaly detection in hardware.

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. This is due to the malicious nature of attacks and intrusions. They 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 of 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 is a problem setting in 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 an unsupervised approach (some kind of explorative data analysis.) 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, SCCH respectively.

- Network traffic (SCCH): The data source is mostly TCP/IP information. This includes the IP addresses, TCP connections, amount of IP packets sent and so. 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.
- Operating system (AIT): The data source is comprised mostly by 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 used in the demonstrators.
- Hardware (TUW): The communication behaviour of particular hardware components together with physical measurements at hardware level is the data pool for this type of analysis. An example of 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 are usually defined as events. 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 and methods in information security.

1.2 Relation to Business Needs and Use Cases

Anomaly detection as outlined above relates to the use cases automated driving (WP6) and smart production (WP7). Requirements are the availability of data. A precise description of the data models that are employed in the 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

In this phase of the project we considered two public datasets for anomaly (attack and intrusion) detection [1-10] with the following goals for research and experimental development:

- NSL-KDD-1999 [9]: Reproduce the known experimental results on this dataset from the last two decades.
- CIC-IDS-2017 [6,8]: Produce novel results for this dataset. This dataset, that describes modern network communications much more realistic than NSL-KDD-1999, has not been considered in detail so far.
- Analytical Toolbox: Develop a user interface for network traffic analysis. As usable security is also a focus of IoT4CPS this should serve as basis for further discussions on how the Analytical Toolbox may be used.

In this phase of the project, we approach anomaly detection by binary classification and supervised learning in general. This means that we assume to have labels for the presence of an attack or intrusion as it is the case with the above mentioned data sets.

2.1 Performance Evaluation of Binary Classifiers

In this section we will have a look on the different metrics and statistics which we will use for model evaluation. The overall goal in binary classification for anomaly detection is to achieve a maximum detection rate and accuracy with a minimum false alarm rate. For conducting our experiments, we implemented the function `evaluation(…)` with the following input arguments:

- `model_fitted`: estimated model which we want to evaluate
- `new_data`: new data on which the model should be evaluated
- `true_labels`: true labels of the new data
- `threshold`: threshold for models returning probabilities (default value is 0.5)
- `print_predicted_labels`: must be set to TRUE if the function should return also the predicted labels for the new data, default setting for this argument is FALSE
- `type`: type of the response for the function `predict()`, this argument is empty by default
- `auc`: must be set to TRUE if the function should return AUC (area under curve) value, default value is FALSE
- `model_is_labels`: must be set to TRUE if the first argument `model_fitted` is already in form of predicted labels, in this case argument `new_data` should be empty

The output of this function is a list of values obtained by various evaluation techniques for binary classification:

- `accuracy`: proportion of correctly classified samples among all samples, i.e. accuracy = (TP + TN)/(TP + TN + FP + FN)
- `TPR` = true positive rate (aka sensitivity or detection rate): proportion of actual positives that are correctly identified as such, i.e. TPR = TP/(TP + FN)
- `TNR` = true negative rate (aka specificity): proportion of actual negatives that are correctly identified as such, i.e. TNR = TN/(TN + FP)
- `balanced_accuracy` = balanced accuracy: (TPR + TNR)/2
- `FAR` = false alarm rate (aka false positive rate): proportion of false positives among all true negatives, FAR = 1 - TNR
- `precision`: proportion of predicted positives from predicted as positive, i.e. precision = TP/(TP + FP)
- `conf_matrix_freq`: confusion matrix with frequencies
- `conf_matrix_perc`: confusion matrix with percentages
- `K` = Cohen's kappa: measures the agreement between two raters who each classify N items into C mutually exclusive categories. If the p-value in the kappa test is lower than predefined level α (e.g. 0.05) then we reject null hypothesis and conclude that the appraiser agreement is significantly different from what would be achieved by chance. This is returned only if the number of different values of predicted labels is equal to number of different values of true labels.

- `predicted_labels`: labels predicted by given model for the given new data. Predicted labels are returned only when `print_predicted_labels` is specified as TRUE in the input.

2.2 The NSL-KDD-1999 Dataset

The data set from the KDD Cup '99 [9] is widely used for evaluation of models for attack and intrusion detection. The data set consists of 41 features and each observation is labeled as either normal (no attack) or with the type of the 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 super user (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, e.g., port scanning

Since the test data is not from the same probability distribution as the train data, the test data consists not only of specific attack types which are not included in train data, but has the new values of the explanatory variables (features) as well. This makes the task more realistic as there are also novelties in test data which needs to be handled.

The KDD Cup '99 [9] data set is rather old and has some deficiencies. The main problem is that the data set consists of repeating observations, thus the classifier constructed by using such a data set would be biased towards more frequent observations. To elude this problem, we used the data set NSL-KDD [9] consisting of unique observations from original KDD Cup '99 [9] data set. This means that the redundant observations are removed and not used in the analysis.

2.2.1 Binary Classification on All Variables

This section presents results from various models, which were used for binary classification of NSL-KDD [9] dataset. The results presented are obtained using all given variables, while the next section presents results using same methods on a subset of the variables - The variables which have near zero variance and which have too high correlation with other variables are removed.

Some of the models required model matrix or one-hot encoded variables, which means that we had to add those values of categorical variables which were only in test data (and not in train data). This is a critical issue that will have to be solved later on in the final version of the tool box.

	Accuracy	TPR	FAR	Precision
Log. reg.	75.61	62.78	7.45	91.77
Bayes log. reg.	75.69	62.93	7.46	91.77
Neural Net	70.54	50.44	2.89	95.84
Naive Bayes	77.70	66.62	7.66	92.00
SVM	77.66	63.08	3.08	96.44
Decision tree	75.84	59.92	3.12	96.21
Ran. Forest	77.54	62.54	2.65	96.90
J48	79.88	68.85	5.53	94.27
knn, k=200	79.17	66.18	3.68	95.97
Keras model	79.13	68.25	6.48	93.30

Figure 1 – Results

Additionally, we trained some neural networks with 3 and 5 hidden layers, with different activation functions and different number of neurons in the hidden layers, but accuracy was every time around 77.5% on the test data. Number of epochs was chosen from the graph of history, not to overfit nor to underfit the model. We used binary cross entropy as a loss function and the optimizer was stochastic gradient descent. There might be a problem with new values of categorical variables in test data, this is taking accuracy lower than where it could be if there were no new categories.

For Keras `nnet`, we configured 3 hidden layers with 100, 80 and 60 neurons with ReLu activation function. The input layer consists of 120 neurons (119 parameters + bias) and output layer consists of only 1 neuron with a sigmoid activation function. The neural network was first trained with 100 epochs and it was validating itself with validation split 0.3. For the purpose of training the batch size was chosen to be 50. This setting gave us an accuracy of 80.6%. Training the model again with the whole training set (no validation split) and with only 40 epochs and the same batch size we get an accuracy on test data of 79.13%. For the purpose of training and testing the standardized data was used.

2.2.2 Binary Classification on a Subset of the Variables

For the next part of experiment, the variables with near zero variance were removed from the test and train datasets. To test if some variable has near zero variance or not, we used the function `nearZeroVariance()` from the R package `Caret`. We removed 22 variables.

	Accuracy	TPR	FAR	Precision
Log. reg.	76.44	64.38	7.63	91.77
Bayes log. reg.	76.47	64.44	7.64	91.77
Neural Net	69.34	50.39	5.61	92.23
Naive Bayes	76.73	64.02	6.48	92.89
J48	79.28	65.99	3.15	96.51
knn, k=200	78.03	63.98	3.40	96.14
Keras model	78.95	66.56	4.68	94.94

Figure 2 – Results (TPR = True Positive Rate, FAR = False Alarm Rate)

As can be noted, the results are almost the same as in the previous experiment, where all variables were used for training models, but the complexity is much lower. Lower training time and better interpretability, together with comparable performance, lead to conclusion that pre-selection of variables based on variance is worth considering.

2.2.3 Multi-Class Classification

In this part of our experiment we tested multi-class classification methods on dataset with four malicious (attack) data types and one non-malicious data type. Four new columns were created for train and test data. Each column is binary and indicates whether a given observation is a specific attack type (DOS, U2R, R2L, and Probe). For example, the binary vector 0110 means that an attack of type U2R and R2L happened. If an observation is not an attack, all 4 values for the given observation are zero.

With this encoding accuracy is good for DOS attacks, all others have a too small amount of observations. New data sets are very imbalanced and thus some classifiers, as for example neural network from the package `nnet`, collapsed and set all predicted values to 0. This gave the best accuracy, but it is very poor predictor.

A neural network trained with the Keras package achieved 78% of accuracy.

2.3 The CIC-IDS-2017 Dataset

For the statistical analysis of intrusion detection, we also used the publicly available dataset CIC-IDS-2017 [6,8]. This dataset consists of network data in the duration of 5 working days. On Monday, there was only 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 the web attacks occurred. On Friday there were some infiltration attacks, as well as some other attacks. Some of the samples which had no label were excluded. Data set is not balanced, there is approximately 80% of benign network traffic and only 20% of malicious network traffic. This is not very good for modelling, but is closer to reality than balanced.

2.3.1 Data Preprocessing

We prepared our data for further analysis as described in the article [1]:

- Remove duplicate column `Fwd Header Length`
- Remove redundant duplicate samples
- Remove the samples with missing label
- Set all missing values and negative values to 0
- Replace infinite values by column mean
- Remove features with zero variance
- Change time formatting
- Add column `hms` for with format `hours:minutes:seconds`
- Add column `binary_attack` if attack is absent or present

We removed features with zero variance: `Bwd PSH Flags, Fwd URG Flags, Bwd URG Flags, CWE Flag Count, Avg Bytes/Bulk, Fwd Avg Packets/Bulk, Fwd Avg Bulk Rate, Bwd Avg Bytes/Bulk, Bwd Avg Packets/Bulk` and `Bwd Avg Bulk Rate.`

Some methods used in our experiment require rescaled data as they are not scale invariant. For this purpose, we scaled our features by a min-max transformation. We replaced a value X by $X' = (X - X_{min}) / (X_{max} - X_{min})$ where X_{max}, X_{min} are the largest and smallest observed value of X. We also used a z-transform, i.e. we replace X by $X' = (X - \text{mean}(X)) / \text{sd}(X)$. Transformation involving subtraction of mean is problematic in our case since all of the variables are originally positive.

For the purpose of dimensionality reduction, we used principal component analysis. This method gives us orthogonal principal components, where each of the components is a linear combination of our features. Principal components are constructed in a way such that the first principal component carries the biggest proportion of variance in the data, the second principal component is orthogonal to the first one and carries the second biggest proportion of the variance in the data, etc.

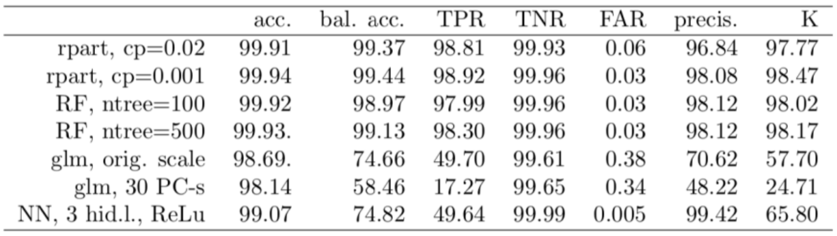
2.3.2 Binary Classification of Brute Force Attack on Tuesday

Brute force attack (password guessing) occurs on Tuesday. The attack is executed with the tools SSH-Patator and FTP-Patator. Both of these attacks last 1 hour. For the purpose of training, we took the first half an hour of both attacks and put them into the Monday training data. Monday was all normal (benign) activity before, now we will train the brute force attacks on it with these two half hours of FTP- and SSH-Patator attacks. Afterwards we used the remaining data of Tuesday for testing of the model. We trained a binary model without timestamps, i.e. with labels of attack/no attack only.

For the purpose of binary classification on the above described dataset we used (non-)parametric methods: Classification trees (R function `rpart`) and an ensemble method of classification trees called Random Forest (R function `randomForest`). We tested a parametric method for binary classification, namely logistic regression (R function `glm`). We used these methods on the original scale (not transformed) and used all of the features.

Additionally, we have transformed the data by min-max transformation and reduced the dimensions by PCA. We have used just some of the principal components and inspected the performance of logistic regression. We have used the data transformed by min-max transformation also to train a neural network with 3 hidden layers with ReLU activation functions and learning rate 0.01 and momentum term 0.5.

All the results in table are in percentages. K is the kappa sample estimate from kappa test done by function `kappa.test`.



	acc.	bal. acc.	TPR	TNR	FAR	precis.	K
rpart, cp=0.02	99.91	99.37	98.81	99.93	0.06	96.84	97.77
rpart, cp=0.001	99.94	99.44	98.92	99.96	0.03	98.08	98.47
RF, ntree=100	99.92	98.97	97.99	99.96	0.03	98.12	98.02
RF, ntree=500	99.93.	99.13	98.30	99.96	0.03	98.12	98.17
glm, orig. scale	98.69.	74.66	49.70	99.61	0.38	70.62	57.70
glm, 30 PC-s	98.14	58.46	17.27	99.65	0.34	48.22	24.71
NN, 3 hid.l., ReLu	99.07	74.82	49.64	99.99	0.005	99.42	65.80

Figure 3 – Results

2.3.3 Other Methods

Logistic regression in both cases (original data and first 30 principal components) did not converge and gave warning that the fitted probabilities numerically 0 or 1 occurred. We have used the model anyway, but we have to keep this warning in mind when comparing the results to other classifiers.

One class support vector machine (scaled data were used) was so slow, that it did not finish computation after a day, so by now we do not have any results. We have tried it only on parts of the data (to have faster classification), but the results were not better than the ones from classification tree described in the beginning of this section. Balanced accuracy was only 59%. This might be misleading as we only have randomly chosen part of the data and there might be too small amount of attacks in it.

Nonparametric k-nearest neighbors did not finish computation after a day as well, so we have no results by now. It is a computationally demanding method and it will presumable not give significantly better result than for example classification tree, which was quite fast.

2.3.4 Multi-Class Classification of Brute Force Attacks on Tuesday

The next part of experiment included testing of multi-class classification methods on the CIC-IDS-2017 dataset. This experiment utilizes the same classifiers as in binary classification experiment, but with 3 different classes: normal data (no-attack), SSH-Patator attack and FTP-Patator attack. The same training and testing subsets as in binary classification were used, with the data transformed by a min-max transformation. The reason for trying to classify each of the two attacks separately is the assumption that they differ in some variables and that this circumstance might have led to such the above result of neural net in binary classification. The results are however almost the same as when we did the binary prediction. The accuracy is 99.07%, balanced accuracy is 74.84%, the TPR is 49.68%, the TNR is 99.99% and the precision is 99.90%.

2.4 Analytical Toolbox

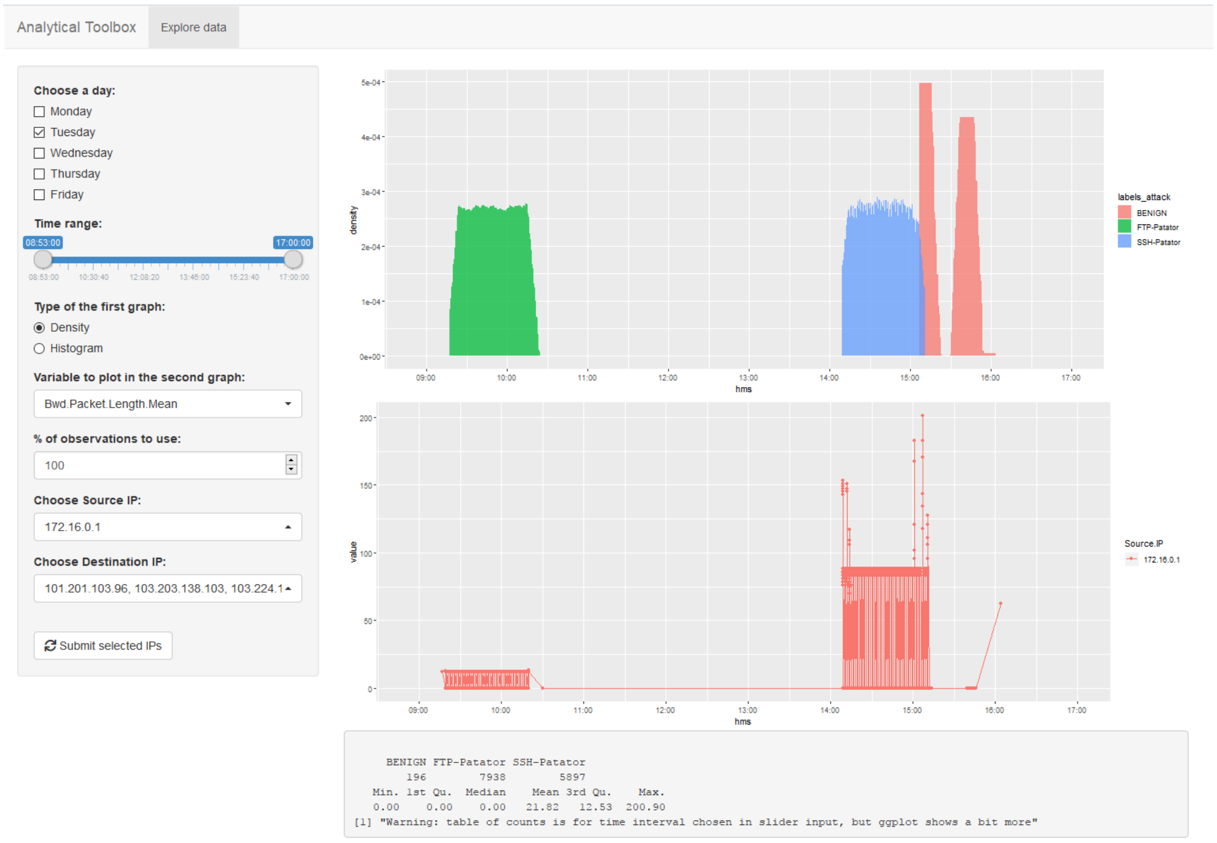
We implemented a so called Shiny-App for visualizing the previous results and allowing for manual (visual) data analysis. Shiny-Apps is a technology for rapid prototyping of applications and based on the statistical programming language R. For demonstration purposes we have a look at the data from Tuesday of the CIC-IDS-2017 [6,8] dataset. On Tuesday there are 2 attacks: FTP-Patator in the morning and SSH-Patator in the afternoon. Both these attacks are coming from a computer with IP 172.16.0.1. This computer produced also benign traffic flow in the afternoon.
Next, we give a brief description of the functionality of the Shiny-App.

Inputs:

- User can choose which day he wants to analyze. Several days can be chosen at once.
- User can choose which time range he wants to analyze. For the chosen time range both graphs will be adjusted.
- User can choose the type of the first (upper) graph to be shown as density or histogram.
- User can choose the variable (feature) to plot in the second graph.
- User can choose a % of observations to use for the plotting (for faster graphics plotting).
- User can choose Source IP and Destination IP of all the flows which he wants to analyze. Both of these options are multiple choice.
- After choosing all desired Source IPs and Destination IPs, one needs to submit the selection to obtain a graph. Shiny will plot all the flows between selected IPs.

Outputs:

- In the first (upper) graph the distribution of labels of the selected observations is shown.
- In the second (lower) graph the line plot of the chosen variable for the chosen Source IP(s) is shown.
- Both graphs can be zoomed by choosing smaller time interval.



Choose a day: Monday [✓ Tuesday] Wednesday Thursday Friday

Time range: 08:53:00 – 17:00:00

Type of the first graph: [✓ Density] Histogram

Variable to plot in the second graph: Bwd.Packet.Length.Mean

% of observations to use: 100

Choose Source IP: 172.16.0.1

Choose Destination IP: 101.201.103.96, 103.203.138.103, 103.224.1

```
   BENIGN FTP-Patator SSH-Patator
      196        7938        5897
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   0.00    0.00    0.00   21.82   12.53  200.90
[1] "Warning: table of counts is for time interval chosen in slider input, but ggplot shows a bit more"
```

Figure 4 – Screenshot of Analytical Toolbox for Anomaly Detection in Network Traffic

Especially in the above figure, we can see the density of samples from the only malign Source IP 172.16.0.1 on Tuesday, with all its flows (all destination IPs are selected). We can see that the variable mean length of backward packets is much higher for SSH-Patator attack (blue) than for FTP-Patator attack (green). There are also some benign observations.

2.5 Summary

To summarize this phase of the project, we conclude that decision tree learning seems to be the best approach for attack and intrusion detection problems (for the given datasets). Random forest learning was a bit slower than the other learning algorithms. Single tree learning also performed good and was quite fast.

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 cyber security 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 described for a blacklist malicious.

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

In addition to the anomaly detection approach described in chapter 0 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, which has been previously developed by AIT[1], is used as a basis for system log analysis tools of the Analytical Toolbox for Anomaly Detection for IoT4CPS. 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.

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

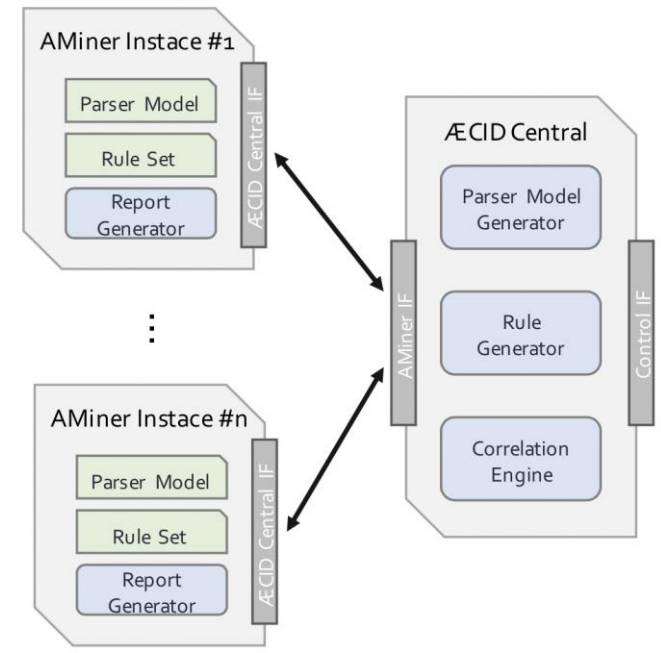


Figure 5 ÆCID architecture

Figure 5 depicts the 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 host and network nodes that needs to be monitored. If available, the AMiner can also be deployed on a 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 parser model[2]. 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 on that system. The parser model in combination with the rule set, describe 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.

ÆCID Central provides 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.

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

Both then can then be used to configure the AMiner instances, in order to detect logged abnormal activities. Additionally, ÆCID Central features a correlation engine which 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 Analytical Toolbox Integration

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

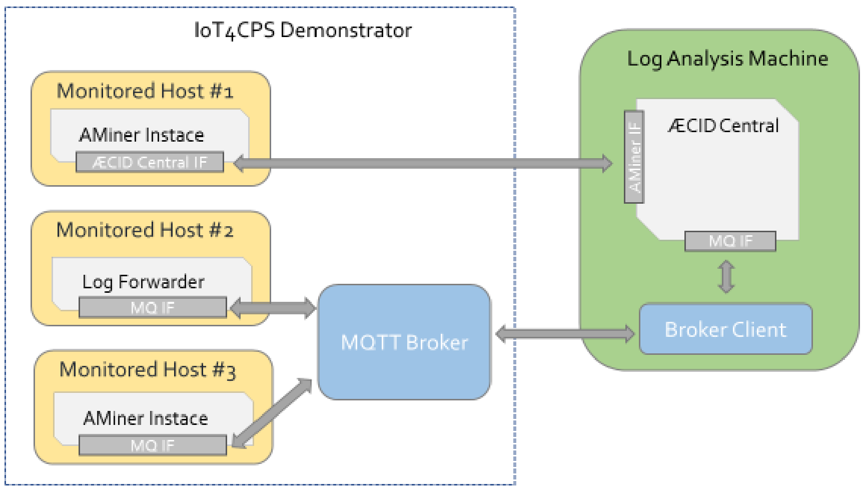


Figure 6 ÆCID integration into IoT4CPS

In order to tackle the first issue, ÆCID needs to be extended with a secure and reliable standardized interface for accessing logs messages of external systems. The choice fell on the use of a message queue, since they allow for asynchronous communication which is ideal for IoT / CPS environments where a constant internet connection of all devices cannot be guaranteed. Message queue provide a temporary message storage until the recipient retrieves them. Apache Kafka[3] 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. Furthermore, solutions for connecting Apache Kafka to a MQTT broker exist.

Figure 6 shows a path of how the integration to the Analytical Toolbox can be accomplished. ÆCID Central can be deployed to an “external” machine, and only exposed through the Apache Kafka Broker and the AMiner Interfaces. In the IoT4CPS demonstrators, three possibilities exist to use the log analysis capabilities of ÆCID:

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

1) Monitored Host #1 shows the standard ÆCID approach, using an AMiner without any special adjustments. Here, only the parser model and a specific set of rules must be deployed to the AMiner instance.
2) Monitored Host #2 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.
3) Monitored Host #3 shows the extended ÆCID approach, featuring an AMiner connected though a MQ interface to the ÆCID Central.

3.4 Summary

To summarize this phase of the project, we have developed a path using the ÆCID framework within the Analytical Toolbox and the IoT4CPS Demonstrators. First steps, the integration of a message queue into ÆCID, have already been undertaken und successfully tested.

4. Hardware Anomaly Detection

Increasing trend of shifting from intelligent edge to intelligent mesh network cyber-physical systems leads to exponential increase in the usage of commercial-off-the-shelf components. The trustworthiness of the these commercial-off-the-shelf components leads to several security threats, especially, the communication network between these components is the most vulnerable toward security attacks at hardware level. Typically, hardware attacks are based on malicious alteration of the hardware design that can alter the functionality, leaks the information or perform the denial-of-service attack (either by inserting the kill-switch [15] or getting the control of controllers [16]).
Several techniques have been proposed to counter these attacks but most of the them are based on the timing, power, current or electromagnetic signals-based golden signatures. However, in the case of commercial-off-the-shelf components, it is nearly impossible to extract the golden signatures. To address this challenge, several analysis techniques have been proposed to analyze the commercial-off-the-shelf components to estimate the golden model under the assumptions that malicious circuitry is not activated during the estimation. This raises the following research challenges: (1) how to ensure the accuracy of the golden circuits? and (2) coverage of the extracted golden circuits. To address these limitations, run-time detection techniques have been proposed, however, most of these techniques are based on the side-channel analysis that require precise calibrations to incorporate the process variations. Moreover, these techniques also premise that triggering of payload results in a substantially higher current flow.

4.1 Statistical Modelling of Communication Network

To address these limitations, we proposed to explore the communication behaviour of because, in real-world scenarios, modules are connected via communication channels and therefore most of the intrusions can have a small or big impact on the communication behaviour. To verify the hypothesis, we analyzed the communication behaviour of an MC8051 microcontroller for the Gaussian and exponential input data distributions, as shown in Fig. 7, and highlighted the following research challenges:

- How to model the communication behaviuor for run-time monitoring with minimum overhead?
- How to exploit the abnormalities in communication behaviuor to design a runtime monitoring framework for HT[4] detection

[4] Malicious hardware intrusions into the integrated circuits, known as Hardware Trojans (HT), can lead to several unwanted payloads, i.e., information leakage, change in the timing characteristics, malfunctioning and denial-of-service (DoS). The effects can be catastrophic, such as system failure and leakage of secret encryption keys (e.g., failure of ice-detection module in the P8A Poseidon [12]).

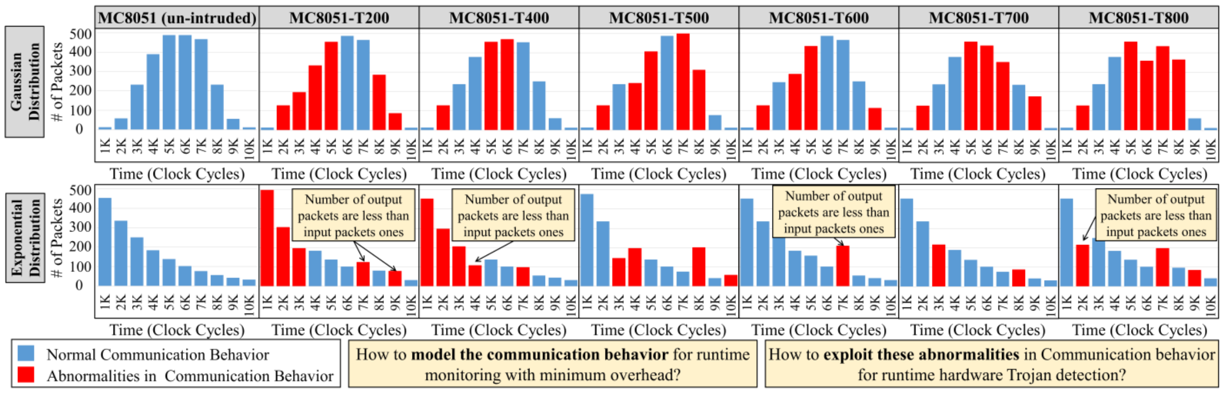


Figure 7: The effects of trust-hub Trojan benchmarks [14] (i.e., MC8051-T200, T300, T400, T500, T600, T700 and T800) on the communication behaviour of MC8051 for Gaussian and Exponential input data distributions [13].

To address the above-mentioned challenges, we propose a runtime anomaly detection methodology based on the traffic modelling of communication in integrated circuits. Using an appropriate threat model is one of the foremost steps in developing any methodology for detecting intrusions in the domain of hardware security. Our approach was published in the [13] we assume that 3PIP vendors are not trusted and the SoC integration is performed in a trusted facility. The proposed methodology consists of two major steps, as shown in Fig. 8:

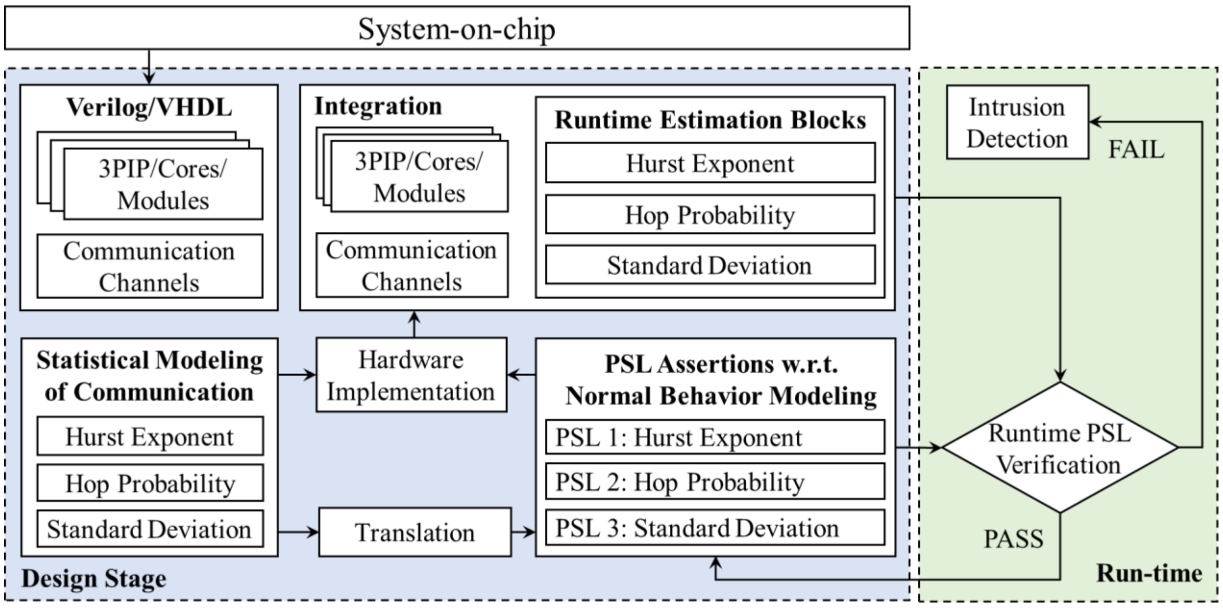


Figure 8: SIMCom: Statistical sniffing of inter-module communication for runtime HT detection

4.1.1 Extract the statistical communication traffic model behaviour

The communication traffic modelling is done under the premise that no Trojans get activated during the pre-market test phase. The statistical communication behaviour of an SoC can be obtained by utilizing the following steps:

- The input packets are generated with respect to the standard spatial injection distribution, e.g., Gaussian distribution.
- Next, Hurst exponent, at the pre-market test stage, is computed for each communication channel.
- Then, the extracted traffic model is translated to its corresponding PSL assertions.

- Finally, the traffic monitoring unit for run-time HT detection is designed. This is achieved by counting the packets for a specific duration and computing the Hurst exponent at run-time.

4.1.2 Run-time Hardware Trojan Detection

The next step of the proposed methodology is to monitor the run-time traffic of an SoC and to compare it with the extracted pre-market test stage behaviour. This step is composed of the following sub-steps:

- First, the Hurst exponent (H)[5], hop probability distribution (P)[6] and standard deviation of input injection distribution (σ) parameters are computed for each communication channel during runtime.
- Then, the extracted statistical communication behaviour is verified using the embedded property specification language (PSL) assertions[7].

4.2 Experimental Analysis

To illustrate the effectiveness of the proposed technique, we implement a network-on-chip (NoC) consists of MC8051 microcontroller that are communicating with each other and UART module via AMB 2.0 bus, as shown in Fig. 9. The main motivation of choosing the MC8051 microcontroller as our case study is the availability of its open-source Trojan benchmarks at the trusthub.org[8]. We first obtained the communication behaviour of the un-intruded MC8051 while communicating with UART modules. We used this behaviour to obtain the corresponding PSL assertions and embed them into the SoC. The key assumption of this analysis is that "Though all instances of the MC8051 can have the Trojan but Trojan in only one IP module is triggered at a particular time considering a unique sequence happening at run time with very low probability of triggering (almost zero).

[5] Hurst Exponent is defined as: $H = \log_{10}\left[\frac{E\left(\frac{R(n)}{S(n)}\right)}{a \times n}\right]$, where, "R(n)" and "S(n)" are the magnitude range and the average magnitude of the time series, "n" is the number of observations and "a" a positive constant.

[6] The hop probability distribution is defined as $P_{h>d} = (1-p)^{s(d)}$, where "s" is the source module, "s(d)" is the number of modules from source with hop distance "d" and "p" is the acceptance probability which is defined as: $p = {}^{1}/_{r-1}$, "r" is the number of receiving modules.

[7] For example, the property, "Hurst exponent a communication channel should be equal to value of the Hurst exponent a communication channel during design stage," is translated into PSL as follows:

PSL 1: `assert always (H[1:100000] == H_design);`

Similarly, for other parameters, we define the following properties:

PSL 2: `assert always (sigma[1:100000] == sigma_design);`

PSL 3: `assert always (P[1:100000] <= Pmax && P[1:100000] >= Pmin)`

[8] Hardware Trojan benchmark available at https://www.trust-hub.org/.

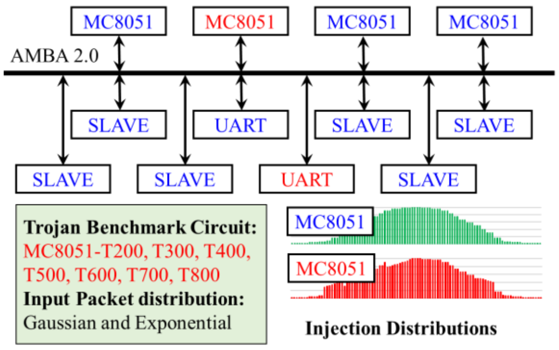


Figure 9: MC8051-based UART Communication Network (blue and red text represent the un-intruded and intruded modules, respectively).

For the comprehensive analysis of SIMCom, we perform the dynamic and average analysis of the statistical communication behaviour over the 100,000 clock cycles, as shown in Fig. 10 and 11.

By analyzing the dynamic communication behaviour (Fig. 10), we made the following key observations:

1. Few of the Trojan impact on Hurst Exponent is dependent on the input data distribution (i.e., Gaussian or Exponential).
2. Hop distribution is useful for such Trojan benchmarks that temporarily or permanently block the communication.
3. All the M8051 Trojan benchmarks have the significant impact on standard deviation of injection distribution of communication data.

By analyzing the average behaviour over 100,000 clock cycles, we made the following key observations:

1. The average behaviour shows that, on the average, all the implemented Trojan benchmarks have a significant impact on Hurst exponent irrespective of the input data distributions, as shown in Fig. 11.
2. In the average-case analysis, all the implemented Trojans have no impact on the probability of hop distribution, except for the MC8051-T300, because it performs the denial-of-service attack and blocks the communication. Trojan benchmark.
3. Similarly, all the implemented Trojans have a significant impact on the standard deviation of the distribution of the injecting traffic (traffic injected by a module in the communication channel).

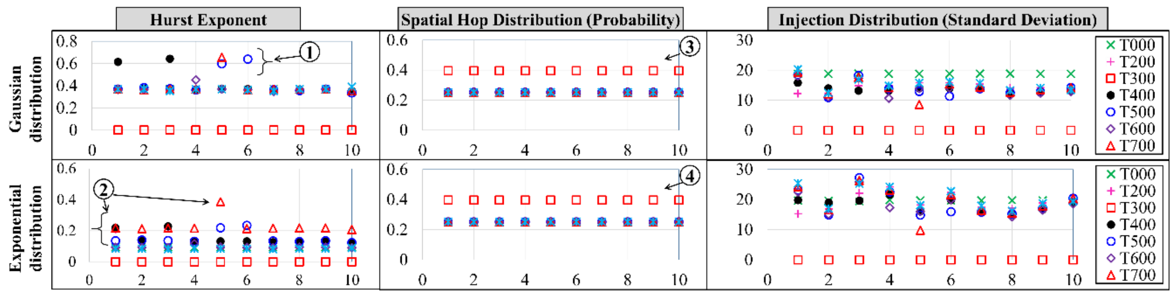


Figure 10: Runtime Impact Analysis of implemented HTs (i.e., MC8051-T200, T300, T400, T500, T600, T700, T800) on the statistical model (Hurst exponent, hop) of the implemented case study for 100,000 clock cycles

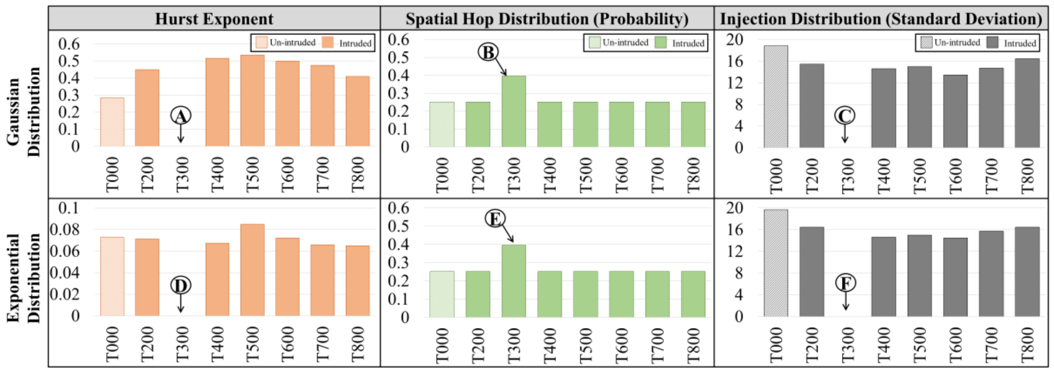


Figure 11: Average Impact Analysis of benchmark Trojans (i.e., MC8051-T200, T300, T400, T500, T600, T700, T800) statistical model of 8051 based UART communication network for 100,000 clock cycles. Note: T000 represents the pre-market test stage communication behavior of the MC8051 based experimental case study with no Trojans inserted. T300 blocks the communication, therefore, it always generates the "0" value for all the statistical parameters except the Hop probability, as shown by the labels A, C, D and F. Hop probability parameter is useful to detect such Trojans that blocks the communication channel.

4.3 Summary

In summary, the proposed statistical model can identify most of the anomalies in communication behaviour due to hardware Trojans. Moreover, unlike the state-of-the-art techniques, it does not require any wrappers and hidden control signals, like the LOCK signal. Moreover, SIMCom can be implemented on any wired or wireless communication system but it possesses the hardware overhead of the online verification and computational modules for estimating the Hurst exponent, hope distribution and standard deviation of the traffic injected by a module in the communication channel.

4.4 Future Work

The SIMCom concept can be extended to Vehicular Ad-Hoc Network, which is emerging as the prominent communication framework for autonomous vehicles. However, due to complex, dynamic and heterogenous communication network topologies, protocols and devices, these networks are vulnerable to several security threats, i.e., information leakage, denial-of-service. In future, we are planning to leverage the statistical modelling of communication pattern (i.e., Hurst Exponent) to generate a single parameter-based unique signature for a particular type of communicating command. Then, these signatures are compared with the run-time communication pattern modelling to identify the anomalous communication behaviour. For illustration, we are planning to implement the proposed approach on the VANET communicating with 802.11p for several datasets, i.e., RioBusses-v2018, VANETjamming2018 and VANETjamming2014 [11].

5. Overall Summary

To summarize this phase of the project, we made progress on anomaly detection within IoT4CPS. Software is under development for the future application of anomaly detection on real-world data. This includes the

- Continuous development of the ÆCID framework within the Analytical Toolbox. In particular, the integration of a message queue into ÆCID has been implemented und successfully tested.
- A data visualization tool that helps in analysing network traffic for detecting attacks and intrusions.
- A statistical model that can identify most of the anomalies in communication behaviour due to hardware Trojans both for wired or wireless communication systems.

In addition, there is experimental software that applies machine learning to public benchmarks for anomaly detection given network traffic data. Our main scientific insight is that decision tree learning seems to be the best approach for attack and intrusion detection problems. The intuitive reason for this is that decision tree learning performs well on event and count data.

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] 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.

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

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

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

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

[12] J. Villasenor and M. Tehranipoor, “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.

[13] Khalid, F., Hasan, S. R., Hasan, O., Awwad, F., & Shafique, M. SIMCom: Statistical Sniffing of Inter-Module Communications for Run-time Hardware Trojan Detection. arXiv preprint arXiv:1901.07299.

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

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

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