



Horizon 2020 Program  
Dynamic countering of cyber-attacks  
SU-ICT-2018



## Cyber security 4.0: Protecting the Industrial Internet of Things

### D3.1: Behavioural analysis and cognitive security framework<sup>†</sup>

**Abstract:** The purpose of this deliverable is to provide a refined specification and description of current version of the Behavioural Analysis & Cognitive Security Framework (BACS) – a framework consisting of behavioural models that enable the analysis of the behaviour of multiple IoT devices. Next, the deliverable describes solutions for building, deploying and managing heterogeneous hybrid cloud environments, with a set of technologies able to handle various platforms. Finally, the deliverable includes description of Intel SGX instructions that BACS behavioural models will use for increased security and privacy-awareness.

Contractual Date of Delivery	31/05/2020
Actual Date of Delivery	31/05/2020
Deliverable Security Class	Public
Editors	<i>Milos Savic, Srdjan Skrbic</i> <i>(University of Novi Sad, Faculty of Sciences)</i>
Contributors	UNSPMF, FORTH, HPE, ITML
Quality Assurance	<i>IBM Israel</i> <i>University of Greenwich</i>

---

<sup>†</sup>The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 833828.

### The *C4IIoT* Consortium

FOUNDATION FOR RESEARCH AND TECHNOLOGY HELLAS	Coordinator	FORTH	EL
CENTRO RICERCA FIAT SCPA	Principal Contractor	CRF	IT
INFINEON TECHNOLOGIES AG	Principal Contractor	IFAG	DE
THALES SIX GTS FRANCE SAS	Principal Contractor	TSG	FR
HEWLETT PACKARD ITALIANA SRL	Principal Contractor	HPE	IT
COMMISSARIAT A L ENERGIE ATOMIQUE ET AUX ENERGIES ALTERNATIVES	Principal Contractor	CEA	FR
IBM ISRAEL - SCIENCE AND TECHNOLOGY LTD	Principal Contractor	IBM	IL
AEGIS IT RESEARCH UG	Principal Contractor	AEGIS	DE
UNIVERSITE PARIS I PANTHEON-SORBONNE	Principal Contractor	UP1PS	FR
INFORMATION TECHNOLOGY FOR MARKET LEADERSHIP	Principal Contractor	ITML	EL
SPHYNX TECHNOLOGY SOLUTIONS AG	Principal Contractor	STS	CH
UNIVERSITY OF NOVI SAD FACULTY OF SCIENCES	Principal Contractor	UNSPMF	SRB
UNIVERSITY OF GREENWICH	Principal Contractor	UOG	UK
VIP MOBILE D.O.O.	Principal Contractor	VIP	SRB

## Document Revisions & Quality Assurance

### Internal Reviewers

1. *IBM Israel*
2. *University of Greenwich*

### Revisions

Version	Date	By	Overview
v3.2 - final	27/05/2020	Srdjan Skrbic (UNSPMF)	Minor edits
v3.1	20/05/2020	Srdjan Skrbic (UNSPMF)	Edited in accordance to review comments
v3.0	04/05/2020	Srdjan Skrbic (UNSPMF)	Edited draft ready for review
v2.0	25/04/2020	Giovanni Giuliani (HPE) Giorgos Vasiliadis (FORTH)	Sections 3 and 4 added
v1.0	20/04/2020	Miloš Savić, Vladimir Jankov, Tamara Krivokuća (UNSPMF)	Section 2 added
v0.0.1	16/03/2020	Srdjan Skrbic (UNSPMF)	ToC

# Table of Contents

<b>1. INTRODUCTION .....</b>	<b>7</b>
<b>2. BEHAVIOURAL ANALYSIS AND COGNITIVE SECURITY MODULE.....</b>	<b>8</b>
2.1 EXPERIMENTAL ANALYSIS OF UNSUPERVISED BACS AD MODELS IN THE SMART FACTORY USE CASE....	14
2.2 EXPERIMENTAL ANALYSIS OF UNSUPERVISED BACS AD MODELS IN THE LOGISTICS USE CASE .....	16
2.3 EXPERIMENTAL ANALYSIS OF SUPERVISED BACS AD MODELS ON NETWORK INSTRUCTION DETECTION DATASETS .....	18
2.4 EXPERIMENTAL ANALYSIS OF UNSUPERVISED BACS AD MODELS ON NETWORK INSTRUCTION DETECTION DATASETS .....	19
<b>3. HETEROGENEOUS HYBRID CLOUD ENVIRONMENT.....</b>	<b>21</b>
3.1 HETEROGENEOUS CLOUD SUPPORT .....	21
3.2 SECURING THE CLOUD LAYER.....	22
3.2.1 <i>Network isolation of C4IIoT modules: K8s Ingress is C4IIoT Cloud Gateway</i> .....	24
3.2.2 <i>Traffic encryption and authentication enforcement</i> .....	25
3.2.3 <i>Cloud Gateway embedded Web Application Firewall</i> .....	26
3.3 ORCHESTRATION OF UTILITY SERVICES .....	28
3.3.1 <i>Certificate Authority services</i> .....	28
3.3.2 <i>Private Attribute Based Encryption Protected Storage</i> .....	29
3.3.3 <i>Private Artifact Repository</i> .....	31
3.3.4 <i>Secure Private Docker image registry</i> .....	31
3.3.5 <i>Infrastructure and component monitoring</i> .....	34
<b>4. RAISING SECURITY AND PRIVACY-AWARENESS USING INTEL SGX .....</b>	<b>36</b>
4.1 INTEL SGX ESSENTIALS .....	36
4.2 METHODOLOGY.....	36
<b>5. HPE INTROSPECT TOOL REPLACEMENT.....</b>	<b>38</b>
<b>6. CONCLUDING REMARKS AND NEXT STEPS .....</b>	<b>39</b>
<b>7. REFERENCES .....</b>	<b>40</b>
<b>APPENDIX - EXPERIMENTAL ANALYSIS OF UNSUPERVISED BACS AD MODELS IN THE SMART FACTORY USE CASE .....</b>	<b>41</b>

## List of Figures

<b>Figure 1.</b> Mapping of BACS packages to the C4IIoT architecture layers. ....	8
<b>Figure 2.</b> Demo showing how to train, evaluate and save an unsupervised BACS AD model. ....	12
<b>Figure 3.</b> Demo showing how to load and use the previously trained unsupervised BACS AD model. ....	12
<b>Figure 4.</b> Demo showing how to train, evaluate and save a supervised BACS AD model. ....	13
<b>Figure 5.</b> Demo showing how to load and use the previously trained supervised BACS AD model. ....	13
<b>Figure 6.</b> Training and inference time depending on the window length for unsupervised BACS AD models on the Device 3 dataset. ....	15
<b>Figure 7.</b> Model size depending on the window length for unsupervised BACS AD models on the Device 3 dataset. ....	15
<b>Figure 8.</b> Training and inference time depending on the training data fraction for unsupervised BACS AD models on the Device 1 dataset. ....	16
<b>Figure 9.</b> Model size depending on window length for unsupervised BACS AD models on the Elevator dataset. ....	16
<b>Figure 10.</b> Training time depending on window length (left) and training data fraction (right) in the logistics use case. ....	17
<b>Figure 11.</b> Model size depending on window length (left) and training data fraction (right) in the logistics use case. ....	17
<b>Figure 12.</b> Inference time depending on window length in the logistics use case. ....	18
<b>Figure 13.</b> Cloud Models Technologies. ....	22
<b>Figure 14.</b> C4IIoT Modules Orchestrator. ....	22
<b>Figure 15.</b> Kubernetes Web-UI interface showing one K8s test namespace. ....	23
<b>Figure 16.</b> Network isolation of C4IIoT modules. ....	24
<b>Figure 17.</b> Mapping rules of incoming URLs and internal (backend) services. ....	25
<b>Figure 18.</b> HTTPS traffic enforcement. ....	25
<b>Figure 19.</b> TLS setup using K8S secrets. ....	26
<b>Figure 20.</b> Cloud gateway web application firewall. ....	27
<b>Figure 21.</b> ModSecurity rules engine enablement. ....	27
<b>Figure 22.</b> EJBCA main administration Web-GUI. ....	28
<b>Figure 23.</b> EJBCA service backend port remapping and TLS support. ....	29
<b>Figure 24.</b> MinIo web GUI. ....	30
<b>Figure 25.</b> K8s ingress configuration focusing MinIO backend port remapping. ....	30
<b>Figure 26.</b> The main page of the Web admin GUI for Nexus 3. ....	31
<b>Figure 27.</b> An overview of a sample image set for “hello-world” (2 tags) pushed on the MiniKube lab deployment of the Harbor system. ....	32
<b>Figure 28.</b> Differences between the two versions of the “hello-world” image. ....	32
<b>Figure 29.</b> The summary of the scan (vulnerability score). ....	33
<b>Figure 30.</b> The Deployment security sections of the configuration. ....	33
<b>Figure 31.</b> Kubernetes cluster monitoring dashboard deployed on the Minikube cluster for C4IIoT. ....	34
<b>Figure 32.</b> A sample view of the state of Prometheus targets on the MiniKube cluster for C4IIoT lab. ....	34
<b>Figure 33.</b> Our approach for privacy-preserving data processing in third-party clouds. ....	37

## List of Abbreviations

<b>AD</b>	Anomaly detection
<b>EE</b>	Elliptic envelope
<b>LOF</b>	Local outlier factor
<b>SVM</b>	Support vector machines
<b>IF</b>	Isolation forest
<b>ABOD</b>	Angle-based outlier detection
<b>KNN</b>	K nearest neighbors
<b>HB</b>	Histogram based
<b>PCA</b>	Principal component analysis
<b>AE</b>	Autoencoder
<b>EUBA</b>	Entity User Behavioural Analytics
<b>UEBA</b>	User Entity Behavioural Analytics

## 1. Introduction

In an effort to respond and mitigate consequences of anomalous behaviour within the C4IIoT system we rely on advanced anomaly detection, as the first step in generating an alarm and compensating and responding to the anomaly. Information is being collected from different inputs and training models is done based on the normal behavior of the system. Anomaly detection within the C4IIoT system is based on parallel and distributed instances of machine learning algorithms implemented at all layers of the C4IIoT architecture (edge node layer, field gateway layer and cloud layer).

Behavioural Analysis & Cognitive Security Framework is a framework that includes behavioural models based on advanced deep learning techniques to provide more contextual information and form an advanced anomaly detection model for the entire IoT ecosystem. The framework takes into account that each device is not isolated and includes its interactions with other devices. Cloud contextual data modeling and appropriate behavioral models enable the aggregation of heterogeneous contextual redundant data and perform complex adaptation processes. The BACS framework is the core of Level-3 security mechanism of the C4IIoT architecture. In the second section, this deliverable contains detailed specification of all components of the framework developed until this moment and reveals future development plans. Moreover, we present detailed results of testing using data that has been developed and acquired within the project.

The architecture of the C4IIoT determines that the cloud layer hosts and manages several modules including mitigation engine, security assurance, as well as behavioural analysis and advanced anomaly detection. The focus of the third section of the deliverable is on resource management and orchestration of the cloud environment. This is in order to achieve provisioning and configuration of the infrastructure resources needed by the cloud-hosted C4IIoT modules, create automated mechanism for seamless integration and configuration of cloud resources and to support building, deploying and managing of the C4IIoT modules inside a hybrid cloud environment.

Finally, the fourth section of the deliverable includes description of Intel SGX instructions that will be used by BACS behavioural models in the future in order to increase security and privacy-awareness even when using third party cloud providers. The technology allows behavioural models to be placed and executed inside private regions of memory called secure enclaves. This type of memory guards our programs even from processes running with administrative privileges. Both code and data residing in an enclave are encrypted in protected areas of operative memory.

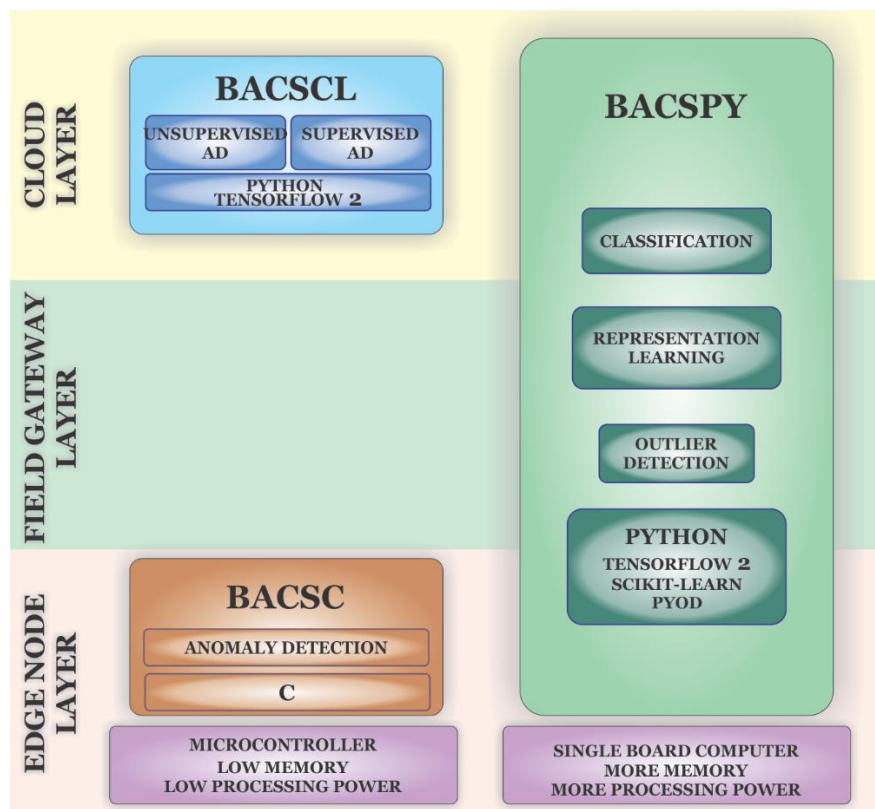
Concluding section provides insights about directions of future development of BACS framework relying on hybrid cloud environment and including Intel SGX technology.

## 2. Behavioural Analysis and Cognitive Security module

BACS (Behavioural Analysis and Cognitive Security) is a C4IIoT software component responsible for detecting anomalies in IIoT sensory data and network traffic flows based on machine/deep learning algorithms. BACS includes both unsupervised anomaly detection (AD) modules detecting deviations from nominal device/network behaviour, as well as supervised AD models trained for specific security threats. The development of BACS started with the C4IIoT project with the goal of providing AD functionalities present at all three layers of the C4IIoT architecture: edge node, field gateway and cloud layer.

BACS consists of the following three main packages (Fig. 1):

1. BACSCL (BACS Cloud Layer) performing anomaly detection based on deep autoencoder forests (unsupervised AD) and deep neural network forests (supervised AD) implemented in Python using the Tensorflow 2 library. BACSCL modules are running in a distributed environment (C4IIoT cloud) and support both data partitioning (training datasets are stored within a distributed file system) and model partitioning (deep autoencoders/neural networks within a forest run on different computational nodes within the cloud). This package is currently in its initial stage of development.



*Figure 1. Mapping of BACS packages to the C4IIoT architecture layers.*

2. BACSPY providing anomaly detection based on outlier detection, classification and representation learning algorithms implemented in Python using Tensorflow 2, scikit-learn and PyOD libraries. BACSPY modules realize AD functionalities running at C4IIoT cloud, field gateways and Raspberry PI based C4IIoT edge nodes devices. This package is in a stable development version. BACSPY supports both standalone and federated AD models. Deep learning based modules from BACSPY package planned for C4IIoT edge nodes in



the smart factory use case are described in Deliverable 2.2. In this deliverable we will describe the rest of the package.

3. BACSC contains lightweight anomaly detection routines implemented in C for constrained microcontroller and based on IIoT devices planned for the logistics use case. Those routines are also described in Deliverable 2.2.

BACSPY package contains the following Python modules:

1. `Dataset` module defines classes representing different types of datasets for training BACS AD models (both unsupervised and supervised).
2. `BaseAD` module defines two abstract classes: `BaseAD` and `BaseSAD`. `BaseAD` is the base class for all BACSPY unsupervised AD models. All BACSPY supervised AD models either directly or indirectly extend `BaseSAD`.
3. `SKLAD` module contains classes realizing unsupervised AD based on outlier detection algorithms implemented in the scikit-learn library.
4. `PyODAD` module contains classes realizing unsupervised AD based on outlier detection algorithms implemented in the PyOD library.
5. `TFAutoAD` module realizes unsupervised AD using Tensorflow autoencoders. The description of this module can be found in Deliverable 2.2.
6. `SKLSAD` module defines classes realizing supervised AD using the the scikit-learn library.
7. `TFSAD` module defines classes realizing supervised AD using Tensorflow2 deep neural networks.

`Dataset` module defines `SWDataset`, `DPDataset` and `LDPDataset` classes. An object of `SWDataset` is a set of multivariate time series constructed using the sliding window approach from time ordered data points. The constructor of the class accepts the name of an input file from which the dataset is formed, the fraction of the set used to train an AD model, window length (the number of data points forming a single time series), time delta (the maximal time difference between the last and the first data point in constructed time series), a boolean value indicating whether the dataset contains GPS coordinates (the default value for this parameter is true) and a boolean value indicating whether GPS coordinates are used as features when training AD models. The class defines the following public methods:

- `get_train_data` returns the training part of the dataset
- `get_train_data_npmatrix` returns the training part of the dataset as a numpy matrix
- `get_test_data` returns the test part of the dataset
- `get_test_data_npmatrix` returns the test part of the dataset as a numpy matrix
- `info` returns the description of the dataset (the number of features and parameters used to construct multivariate time series).

`DPDataset` represents a set of individual data points (time series of length 1), while `LDPDataset` is a set of labelled data points that can be used to train supervised AD models. Public methods of `DPDataset` and `LDPDataset` classes are the same as in `SWDataset` class.

`BaseAD` class is the base class for all unsupervised BACS models. The constructor of the class accepts the following parameters: model name (string), anomaly detection method (an object responding to `fit` method that is called to train the model from a numpy matrix), and dataset

(an object of `SWDataset` containing the training data). The most important methods defined in the class are the following:

- `train` – an abstract method whose derived implementations train an unsupervised AD model (Figure 2)
- `ad_inference` – an abstract method whose derived implementations perform anomaly detection on a given sample (time series) returning anomaly detection confidence scores in the interval  $[0, 1]$
- `is_anomaly` – a method performing anomaly detection on a given sample returning a tuple containing a boolean value (true if anomaly detected, false otherwise) and the normalized anomaly detection confidence score (Figures 2-3)

The constructor of `BaseSAD` class (the base class for supervised BACS anomaly detection models) has two parameters: model name and dataset (an object of `LPDataset` containing the training data). The most important methods defined in the class are the following:

- `train` – an abstract method whose derived implementations train a supervised AD model (Figure 4)
- `sad_inference` – an abstract method whose derived implementations perform supervised anomaly detection on a given sample returning normalized confidence scores
- `is_anomaly` – a method performing anomaly detection on a given sample returning a tuple containing a boolean value (true if anomaly detected, false otherwise), anomaly type and the normalized anomaly detection confidence score (Figures 4-5)
- `evaluate` – a method which computes accuracy, precision and recall scores on the test data (Figure 4)

Besides `BaseAD` and `BaseSAD` classes, `BaseAD` module also defines functions for saving and loading BACS AD models (Figures 2-5).

`SKLAD` module defines a class of the same name extending `BaseAD`. `SKLAD` class is the base class for all classes realizing unsupervised anomaly detection relying on outlier detection algorithms implemented in the scikit-learn library. The following classes extend `SKLAD`:

1. `EE_SKLAD` – unsupervised AD based on the elliptic envelope algorithm
2. `SVM_SKLAD` – unsupervised AD based on one-class support vector machines
3. `LOF_SKLAD` – unsupervised AD based on the local outlier factor algorithm
4. `IF_SKLAD` – unsupervised AD based on isolation forests

`PyODAD` module contains a base class for classes using outlier detection methods from the `PyOD` library. This module implements the following classes:

1. `ABOD_PyODAD` – unsupervised AD realized using the angle-based outlier detection algorithm
2. `KNN_PyODAD` – unsupervised AD based on the k nearest neighbours algorithm
3. `PCA_PyODAD` – unsupervised AD based on principal component analysis
4. `HBO_PyODAD` – unsupervised AD realized using the histogram-based outlier detection algorithm

Figure 2 shows a simple program demonstrating how to instantiate, train and evaluate one of the unsupervised BACS AD models. In the example, `SVM_SKLAD` model is trained on a dataset collected from a C4IIoT edge node device fabricated for the logistics use case. The

example straightforwardly generalizes to any other unsupervised BACS AD model. Figure 3 shows how to load the previously trained model and use it to perform anomaly detection inference.

SKLSAD module contains classes for supervised AD based on the scikit-learn library. This module defines the following classes:

1. SVM\_SKLAD – supervised AD based on support vector machines
2. RF\_SKLAD – supervised AD based on random forests
3. NB\_SKLAD – supervised AD based on the naive Bayes algorithm
4. KNN\_SKLAD – supervised AD based on the K nearest neighbours algorithm

TFSAD module defines two classes for supervised anomaly detection based on Tensorflow2 deep neural networks:

1. TFBNNNSAD – a deep neural network performing binary classification
2. TFNNNSAD – a deep neural network performing  $n$ -ary classification

A simple demo showing how to train, evaluate and save a supervised BACS AD model is shown in Figure 4. In this example, RF\_SKLAD model is trained to recognize DDoS network traffic flows using the CICDS2017 network intrusion detection dataset [1]. The example shown in Figure 5 demonstrates how to load and use the previously trained model.

```
from BACSPY.Dataset import SWDataset
from BACSPY.SKLAB import SVM_SKLAD
from BACSPY.BaseAD import save_model

# step 1 -- make the dataset from the input file
dataset = SWDataset("devlog.csv", train_fraction = 0.7,\
    window_length = 5, use_location_for_ad = True)
dataset.info()
print("\n")

# step 2 -- train the model
model = SVM_SKLAD(dataset)
model.train()

# step 3 -- examine the model
num_anomalies = 0
for t in dataset.get_test_data():
    anomaly_detected, confidence = model.is_anomaly(t)
    if anomaly_detected:
        print("Anomaly detected, confidence = ", confidence)
        num_anomalies += 1
print(num_anomalies, " anomalies detected")

# step 4 -- save the model
save_model(model, "svm.model")
```

```
svc@svcpc:~/BACS_wrk/python/demoAD$ python3 demo1.py
num features = 11
features = ['timestamp', 'GPS_lat', 'GPS_lon', 'GPS_alt', 'GPS_speed', 'GPS_num_sats',
'ACC_x_RMS', 'ACC_y_RMS', 'ACC_z_RMS', 'ACC_samples', 'MAG_x_RMS', 'MAG_y_RMS', 'MAG_
z_RMS', 'MAG_samples']
window length = 5

0 anomalies detected
Saving model to svm.model
svc@svcpc:~/BACS_wrk/python/demoAD$ █
```

**Figure 2.** Demo showing how to train, evaluate and save an unsupervised BACS AD model.

```
import numpy as np
from BACSPY.SKLDAD import SVM_SKLDAD
from BACSPY.BaseAD import load_model

# step 1 -- load the model
model = load_model("svm.model")

# step 2 -- make prediction (inference) on a random time series
rnd_vector = np.random.uniform(low = 0, high = 100, size = 55).tolist()
anomaly_detected, confidence = model.is_anomaly(rnd_vector)
if anomaly_detected:
    print("Anomaly detected, conf = ", confidence)
```

```
svc@svcp:~/BACS_wrk/python/demoAD$ python3 demo2.py
Loading model from svm.model
Anomaly detected, conf = 0.999999999911291
svc@svcp:~/BACS_wrk/python/demoAD$ █
```

**Figure 3.** Demo showing how to load and use the previously trained unsupervised BACS AD model.

```
import numpy
from BACSPY.Dataset import LDPDataset
from BACSPY.SKLSAD import RF_SKLSAD
from BACSPY.BaseAD import save_model

# step 1 -- make the dataset from the input file
print("Loading DDos dataset...")
data = LDPDataset("DDos_ISCX.csv", train_fraction = 0.8,\
    label_field_name = "Label", regular_data_labels = ["BENIGN"])
data.info()

# step 2 -- train the model
print("\nTraining the random forest model")
model = RF_SKLSAD(data)
model.train()

# step 3 -- evaluate the model
print("\nModel evaluation")
model.evaluate()

# step 4 -- save the model
save_model(model, "rf_ddos_model")
```

```

svc@svcp:~/BACS_wrk/python/demoSAD$ python3 demo1SAD.py
Loading DDoS dataset...
num data points = 225711
num features = 78
num classes = 2
features = ['Destination Port' 'Flow Duration' 'Total Fwd Packets'
'Total Backward Packets' 'Total Length of Fwd Packets'
'Total Length of Bwd Packets' 'Fwd Packet Length Max'
'Fwd Packet Length Min' 'Fwd Packet Length Mean'
'Fwd Packet Length Std' 'Bwd Packet Length Max' 'Bwd Packet Length Min'
'Bwd Packet Length Mean' 'Bwd Packet Length Std' 'Flow Bytes/s'
'Flow Packets/s' 'Flow IAT Mean' 'Flow IAT Std' 'Flow IAT Max'
'Flow IAT Min' 'Fwd IAT Total' 'Fwd IAT Mean' 'Fwd IAT Std'
'Fwd IAT Max' 'Fwd IAT Min' 'Bwd IAT Total' 'Bwd IAT Mean'
'Bwd IAT Std' 'Bwd IAT Max' 'Bwd IAT Min' 'Fwd PSH Flags'
'Bwd PSH Flags' 'Fwd URG Flags' 'Bwd URG Flags' 'Fwd Header Length'
'Bwd Header Length' 'Fwd Packets/s' 'Bwd Packets/s'
'Min Packet Length' 'Max Packet Length' 'Packet Length Mean'
'Packet Length Std' 'Packet Length Variance' 'FIN Flag Count'
'SYN Flag Count' 'RST Flag Count' 'PSH Flag Count' 'ACK Flag Count'
'URG Flag Count' 'CWE Flag Count' 'ECE Flag Count' 'Down/Up Ratio'
'Average Packet Size' 'Avg Fwd Segment Size' 'Avg Bwd Segment Size'
'Fwd Header Length.1' 'Fwd Avg Bytes/Bulk' 'Fwd Avg Packets/Bulk'
'Fwd Avg Bulk Rate' 'Bwd Avg Bytes/Bulk' 'Bwd Avg Packets/Bulk'
'Bwd Avg Bulk Rate' 'Subflow Fwd Packets' 'Subflow Fwd Bytes'
'Subflow Bwd Packets' 'Subflow Bwd Bytes' 'Init Win bytes forward'
'Init Win bytes backward' 'act data pkt fwd' 'min seg size forward'
'Active Mean' 'Active Std' 'Active Max' 'Active Min' 'Idle Mean'
'Idle Std' 'Idle Max' 'Idle Min' 'Label']
classes = {'DDoS', 'BENIGN'}
regular data labels = ['BENIGN']

Training the random forest model

Model evaluation
Labels = {'DDoS', 'BENIGN'}
Accuracy 0.9997064833941596
Precision [0.99993169 0.99941101]
Recall [0.99955125 0.99991033]

Saving model to rf_ddos_model
svc@svcp:~/BACS_wrk/python/demoSAD$ █

```

**Figure 4.** Demo showing how to train, evaluate and save a supervised BACS AD model.

```

import numpy

from BACSPY.SKLSAD import RF_SKLSAD
from BACSPY.BaseAD import load_model

# step 1 -- load the model
model = load_model("rf_ddos_model")

# step 2 -- make prediction (inference) on a random network traffic
rnd_vector = numpy.random.uniform(low = 0, high = 100,\
    size = 78).tolist()
anomaly_detected, predicted_type, conf = model.is_anomaly(rnd_vector)
print("Anomaly detected = ", anomaly_detected)
print("Predicted type    = ", predicted_type)
print("Confidence        = ", conf)

```

```

svc@svcp:~/BACS_wrk/python/demoSAD$ python3 demo2SAD.py
Loading model from rf_ddos_model
Anomaly detected = False
Predicted type   = BENIGN
Confidence       = 0.9
svc@svcp:~/BACS_wrk/python/demoSAD$ █

```

**Figure 5.** Demo showing how to load and use the previously trained supervised BACS AD model.

## 2.1

### 2.1 Experimental analysis of unsupervised BACS AD models in the smart factory use case

In this Section we present results of an initial analysis of BACS unsupervised models in the smart factory use case. The analysis was performed on datasets constructed from a log file containing sensory measurements captured by a sensory system deployed in the CRF factory in a period of one day (5.11.2019). The sensory system consists of 9 heterogeneous devices having 149 sensors in total. Those sensors are taking various kinds of measurements from temperature and pressure to speed and acceleration of different moving parts of the system. For each sensor, the logged data is aggregated at the interval of one minute, that is, every minute for which measurements were recorded, and four statistics measurements are computed: minimal value, maximal value, mean value and standard deviation of all measurements occurring within that minute. This way key information is preserved while optimizing the number of data points. In our future work we will consider also other data aggregation scales (e.g. 30 seconds or 10 minutes).

We have analyzed how scalability properties of unsupervised BACS AD models (training time, inference time and model size) are affected by the window length parameter (the size of time series on which anomaly detection inference is performed) and the size of the training dataset. More concretely, it was examined how

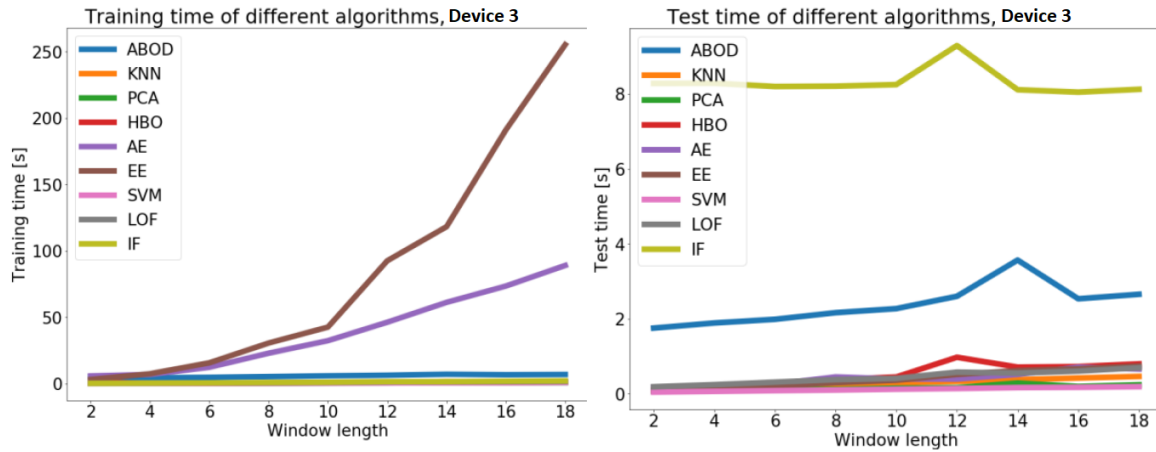
- training time changes with training dataset size and sliding window length,
- inference time changes with sliding window length, and
- model size changes with training dataset size and sliding window length.

Code for the conducted experiments was written in Python 3.7 programming language. The experiments were performed on: Intel Core i5-8250U, 8GB DDR4 (experiments in which we analyzed the impact of the sliding window length) and AMD Radeon R9 270, 8GB DDR4 (experiments in which the training dataset size was varied).

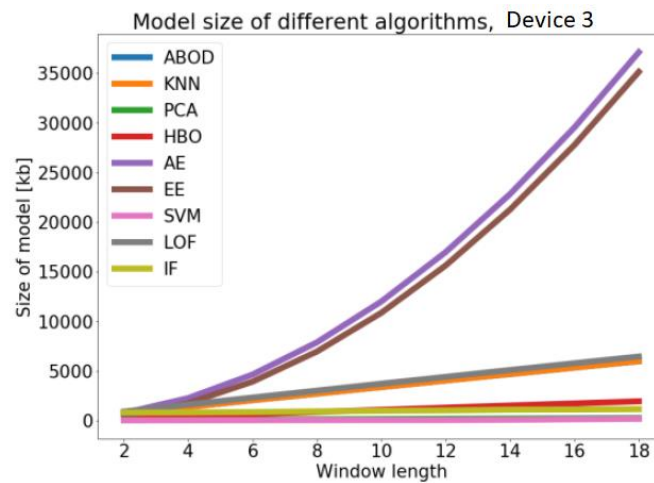
As previously mentioned, there are nine devices whose behaviour was analysed. Some of these devices behave similarly, so results will not be discussed in detail for all of them. All graphs obtained in this analysis are given in Appendix A.

**Sliding window length.** Certain devices have less than 300 data points (before data aggregation), meaning that choosing large window lengths would reduce the number of data points a lot and make already the large number of features much larger. A large majority of devices have more data points but also a large number of features. For the window length parameter varied in the interval [2, 20] we have obtained the following results:

- *Training time.* Nearly all of the unsupervised BACS AD models exhibit the same behavior for different device datasets. The training time of AE and EE significantly increases with the sliding window length. For other models the training time is not significantly affected by this parameter (Figure 6, left).
- *Inference time.* The models exhibiting significant inference time are IF (up to 9 seconds) and ABOD (up to 4 seconds). Anomaly detection performed by other models takes less than a second (Figure 6, right).
- *Model size.* The size of AE and EE models significantly grows with the sliding window length. For other models we have either a small linear growth or constant model size (Figure 7).



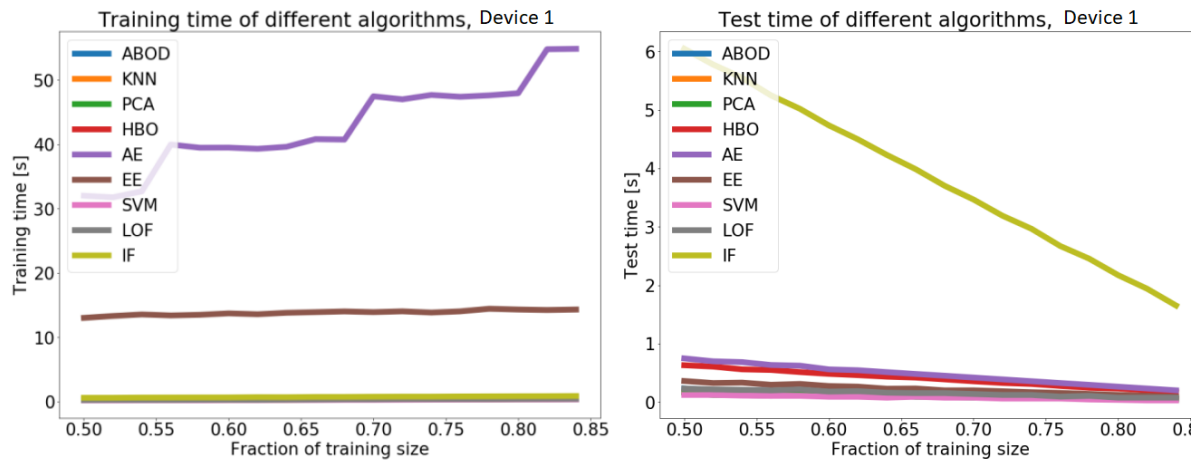
**Figure 6.** Training and inference time depending on the window length for unsupervised BACS AD models on the Device 3 dataset.



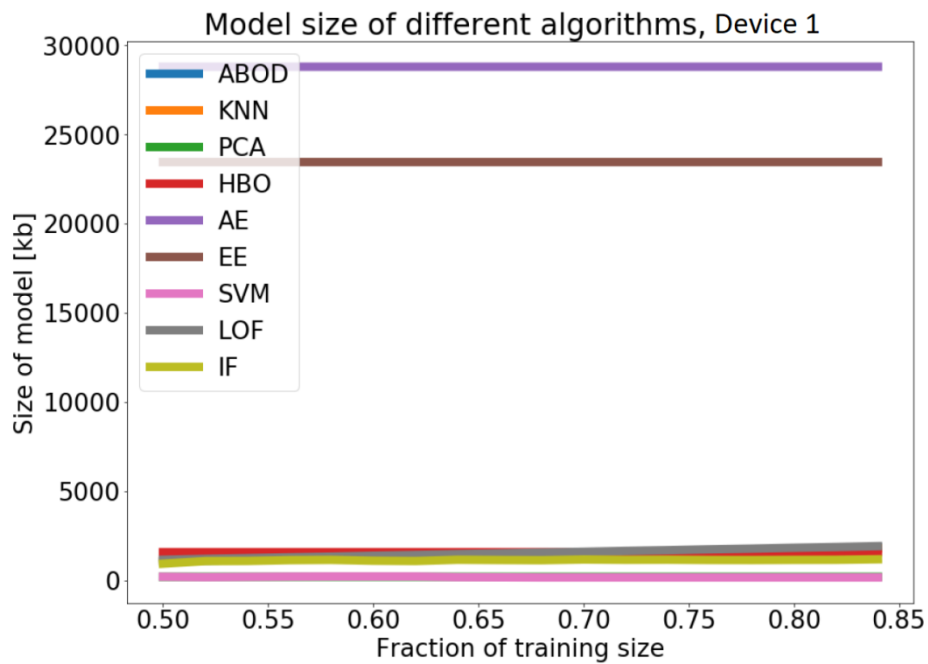
**Figure 7.** Model size depending on the window length for unsupervised BACS AD models on the Device 3 dataset.

**Training dataset size.** As mentioned earlier, the training data fraction parameter determines the percentage of the logged data included in the training of the models. We have varied this parameter from 50% to 85% and obtained the following results:

- *Training time.* Similarly as for the window length parameter, AE and EE models have the largest training time that is significantly increasing with the size of the training dataset (Figure 8 left).
- *Inference time.* IF is the only algorithm that exhibits a large inference time (up to 6 seconds, Figure 8 right); all other models make inference in less than a second. As expected, the inference time declines with the training data size (since the test dataset becomes smaller through this process). Figure 8 (right) shows the inference time cumulatively, enabling us to observe methods having comparatively larger inference times. In our future analyses we will also consider other approaches to examine the influence of increasingly larger training sets over the inference time (e.g., using a fixed test set to measure the inference time for increasingly larger training sets).
- *Model size.* In most cases, AE and EE recorded the largest model sizes. However, no significant increase in model size was observed (Figure 9).



**Figure 8.** Training and inference time depending on the training data fraction for unsupervised BACS AD models on the Device 1 dataset.



**Figure 9.** Model size depending on window length for unsupervised BACS AD models on the Elevator dataset.

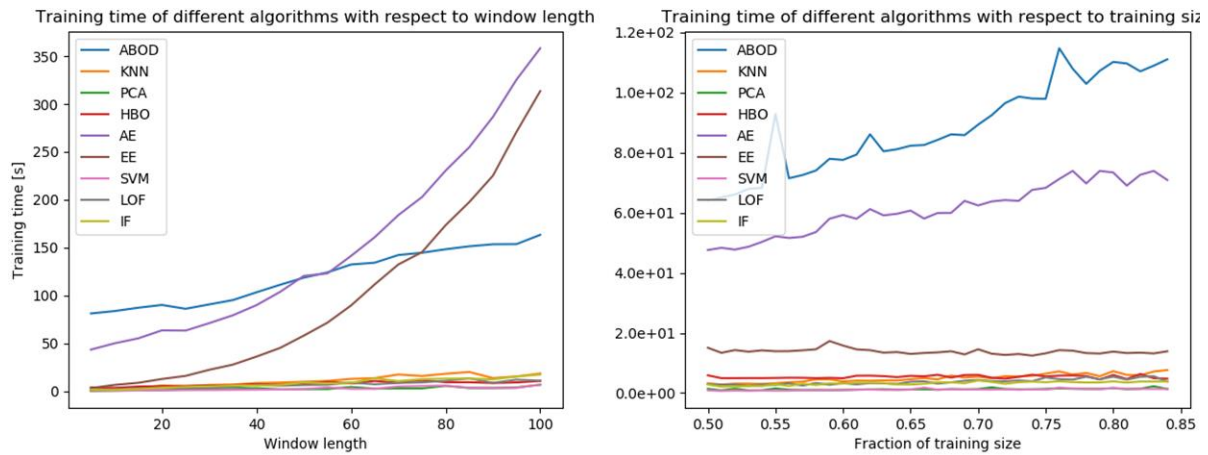
## 2.2 Experimental analysis of unsupervised BACS AD models in the logistics use case

Using the identical experimental methodology as in the previous Section we have made an initial analysis of unsupervised BACS AD models in the logistics use case. The analysis is based on a dataset containing sensory data from a NB-IoT device capturing environmental conditions (temperature, humidity, pressure and lux; 10 features in total). The experimental dataset consists of 5879 data points.

Figure 10 shows how the training time changes with the window length and training data fraction. It can be observed that ABOD, AE and EE are models requiring significant training time that is rapidly increasing with the window length parameter. Additionally, the training size

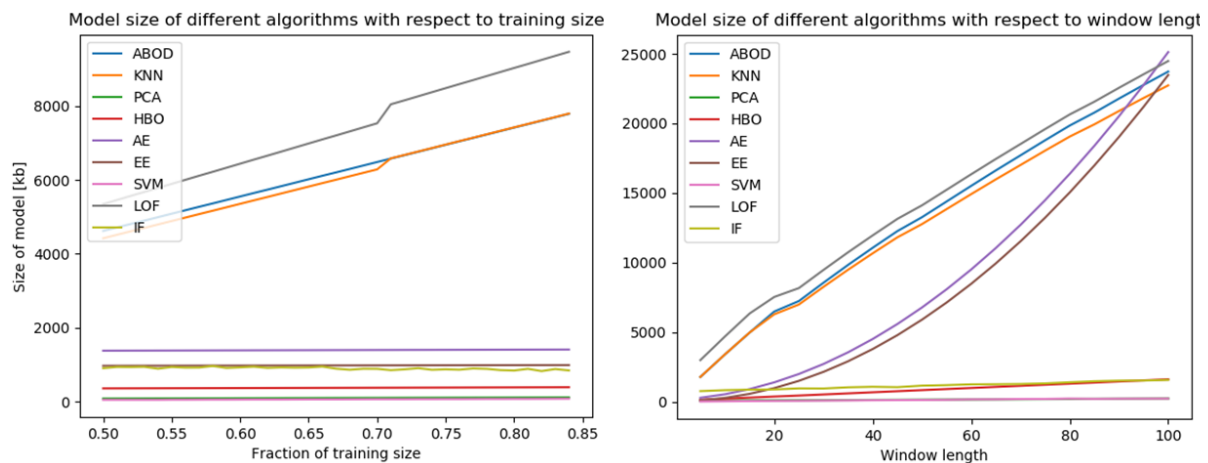


of ABOD and AE steadily grows with training dataset size, which is not the case for the rest of the models.



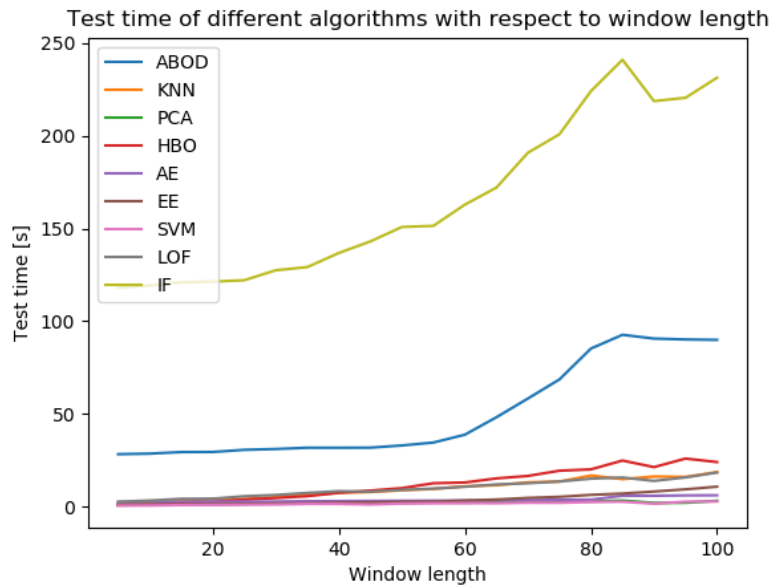
**Figure 10.** Training time depending on window length (left) and training data fraction (right) in the logistics use case.

The impact of window length and training dataset size on the size of BACS models is shown in Figure 11. The size of ABOD, KNN, LOF, AE and EE considerably increases with the window length. Additionally, the training dataset size significantly affects the size of ABOD, KNN and LOF models.



**Figure 11.** Model size depending on window length (left) and training data fraction (right) in the logistics use case.

Figure 12 shows how the window length parameter impacts the anomaly detection inference time for different unsupervised BACS AD models. It can be seen that IF and ABOD have high inference times. Additionally, the inference time of IF and ABOD increases at a faster rate with the window length compared to other models.



**Figure 12.** Inference time depending on window length in the logistics use case.

### 2.3 Experimental analysis of supervised BACS AD models on network intrusion detection datasets

Supervised BACS AD models were analytically examined on two CICDDS2017 network intrusion detection datasets [1]. The first dataset combines DDoS network traffic flows with normal network traffic. In the second dataset normal network traffic is mixed with PortScan attacks. The features present in both datasets can be seen in Figure 2. The 80-20% split into the training and test set was used to evaluate supervised BACS AD models for both attacks. We have computed the following evaluation measures:

- Accuracy – the number of correctly classified data points divided by the total number of data points in the test set.
- Precision for benign data points (P-benign) – the number of correctly identified benign data points (normal traffic flow) divided by the number of data points in the test set that a supervised AD method identified as benign.
- Recall for benign data points (R-benign) – the number of correctly identified benign data points divided by the total number of benign data points in the test set.
- Precision for malicious data points (P-ddos/P-pscan) – the number of correctly identified attacks divided by the number of times when an AD method alarmed an attack.
- Recall for malicious data points (R-ddos/R-pscan) – the number of correctly identified attacks divided by the total number of attacks present in the test set.

The obtained results are summarized in Table 1. It can be seen that AD based on random forests, KNN and deep learning methods (TFNNSAD and TFBNNSAD) for both attack types exhibits 99% accuracy with precision and recall values that are also equal to 0.99 for both benign and malicious network flows. On the other hand, SVM and Naive Bayes have significantly lower accuracy, precision and recall scores.

<b>DDOS</b>	<b>Accuracy</b>	<b>P-benign</b>	<b>R-benign</b>	<b>P-ddos</b>	<b>R-ddos</b>
RF_SKLAD	0.99	0.99	0.99	0.99	0.99
SVM_SKLAD	0.92	0.92	0.9	0.93	0.94
KNN_SKLAD	0.99	0.99	0.98	0.98	0.99
NB_SKLAD	0.84	0.99	0.63	0.78	0.99
TFNNNSAD	0.99	0.99	0.99	0.99	0.99
TFBNNSAD	0.99	0.99	0.99	0.99	0.99

<b>PortScan</b>	<b>Accuracy</b>	<b>P-benign</b>	<b>R-benign</b>	<b>P-pscan</b>	<b>R-pscan</b>
RF_SKLAD	0.99	0.99	0.99	0.99	0.99
SVM_SKLAD	0.69	0.96	0.32	0.64	0.99
KNN_SKLAD	0.99	0.99	0.99	0.99	0.99
NB_SKLAD	0.74	0.98	0.42	0.68	0.99
TFNNNSAD	0.99	0.99	0.99	0.99	0.99
TFBNNSAD	0.99	0.99	0.99	0.99	0.99

**Table 1.** Evaluation of supervised BACS AD models on intrusion detection datasets

## 2.4 Experimental analysis of unsupervised BACS AD models on network instruction detection datasets

Experimental analysis of unsupervised BACS AD models on network instruction detection datasets [1] was done in two scenarios:

- Experiment 1. The datasets were divided into the training and test part in the way that the training part contains all benign data points (normal network traffic), while the test part encompasses all malicious data points (DDoS/PortScan network traffic). In other words, in this experiment we have evaluated unsupervised BACS AD models only against malicious network traffic.
- Experiment 2. In this experiment, the 80-20% train-test split was done for both datasets. Then from the training part we have eliminated malicious data points, so the unsupervised BACS models were trained only with benign data points, but evaluated against both normal and malicious network traffic.

In both experiments we have computed the accuracy of trained models. In the first experiment, the accuracy of a model corresponds to the number of correctly identified anomalies divided by the size of the test set. In the second case, the accuracy is equivalent to  $(TP + TN) / S$ , where  $S$  is the size of the test set,  $TP$  is the number of true positives (correctly identified anomalies) and  $TN$  is the number of true negatives (correctly identified non-anomalies).

The results of both experiments are summarized in Table 2. It can be observed that in both experiments on both experimental datasets one-class support vector machines achieve high accuracy values (higher than 0.85) indicating that this particular unsupervised model is able to detect malicious network traffic. For other models we obtained considerably smaller accuracy values. However, it should be emphasized that both experiments were done using default values for hyper parameters of examined models. Therefore, in our future experiments we will employ different hyper parameter tuning procedures in order to find the best configurations of hyper parameters for individual models.

<b>Exp 1.</b>	<b>EE</b>	<b>SVM</b>	<b>LOF</b>	<b>IF</b>	<b>KNN</b>	<b>PCA</b>	<b>HBO</b>	<b>ABOD</b>
DDoS	0.3826	0.8672	0.6672	0.2693	0.6343	0.6327	0.1648	0
PortScan	0.138	0.8623	0.1376	0.1375	0.1374	0.0065	0.0004	0
<b>Exp 2.</b>	<b>EE</b>	<b>SVM</b>	<b>LOF</b>	<b>IF</b>	<b>KNN</b>	<b>PCA</b>	<b>HBO</b>	<b>ABOD</b>
DDoS	0.406	0.8678	0.6588	0.2754	0.6454	0.6682	0.2618	0.1321
PortScan	0.1246	0.862	0.5969	0.5568	0.1261	0.1296	0.1224	0.1376

**Table 2.** Evaluation of unsupervised BACS AD models on intrusion detection datasets

### 3. Heterogeneous Hybrid Cloud Environment

The Cloud Layer of the C4IIoT architecture is hosting and orchestrating inside a “cloud” several modules, e.g. Mitigation Engine, Security Assurance, ML-based Behavioural Analysis, Advanced Anomaly detection, etc.

The focus of Task 3.1 is in fact the “Resource Management and Orchestration” of the cloud environment to achieve the following goals:

- Provision and configure the infrastructure resources needed by the cloud-hosted C4IIoT modules
- Create automated mechanism to seamlessly integrate and configure cloud resources
- Support building, deploying and managing C4IIoT modules inside a hybrid cloud environment

All these tasks also require collaboration with Task 4.3 and trial use cases to create a consistent environment.

#### 3.1 Heterogeneous cloud support

To achieve the widest usability of C4IIoT modules, they need to be portable across a large variety of different “cloud models”, and at the same time be able to support various “cloud technologies”.

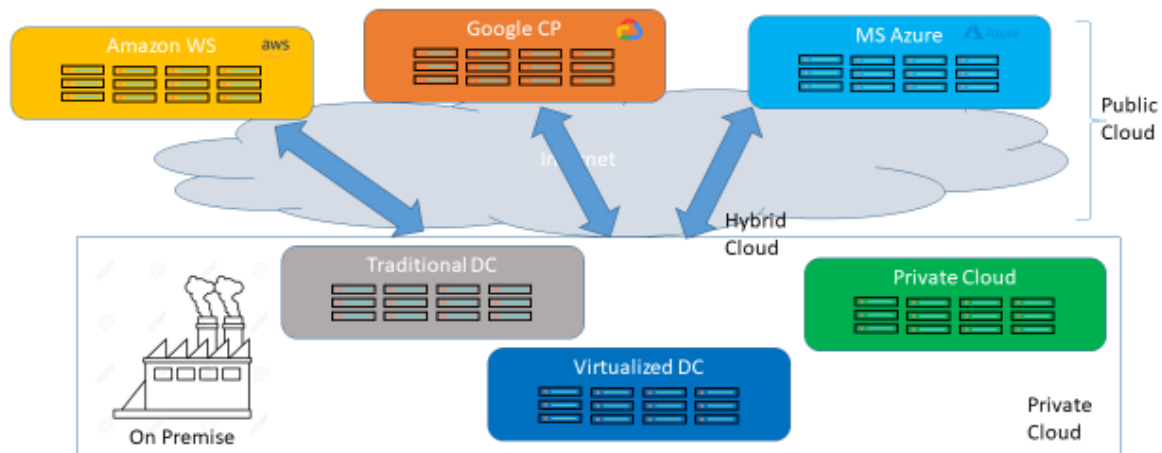
By “cloud models” we mean Public Cloud (resources completely hosted by external provider), Private (resources are all on-premise) or Hybrid (mix of on premise and remote resources). Examples of popular “public cloud technologies” are Amazon Web Services (AWS) [1], Google Cloud Platform [2] and Microsoft Azure [3]; and there’re Open Source technologies like OpenStack [4], frequently used in Private Clouds. For C4IIoT project we want to include both the traditional Data Center technologies like Virtualization and managed physical servers, as summarized in the Figure 13.

To satisfy this requirement, all Cloud Layer C4IIoT components will be built as independent Docker Containers to be deployed through a cloud-agnostic orchestrator, which is available on a large variety of Cloud Platforms – of different models and using various technologies.

The choice for C4IIoT orchestrator has been for Kubernetes (<https://kubernetes.io/>) that offers an “open-source system for automating deployment, scaling, and management of containerized applications” and it’s widely used on most cloud platforms and has a very active open-source community.

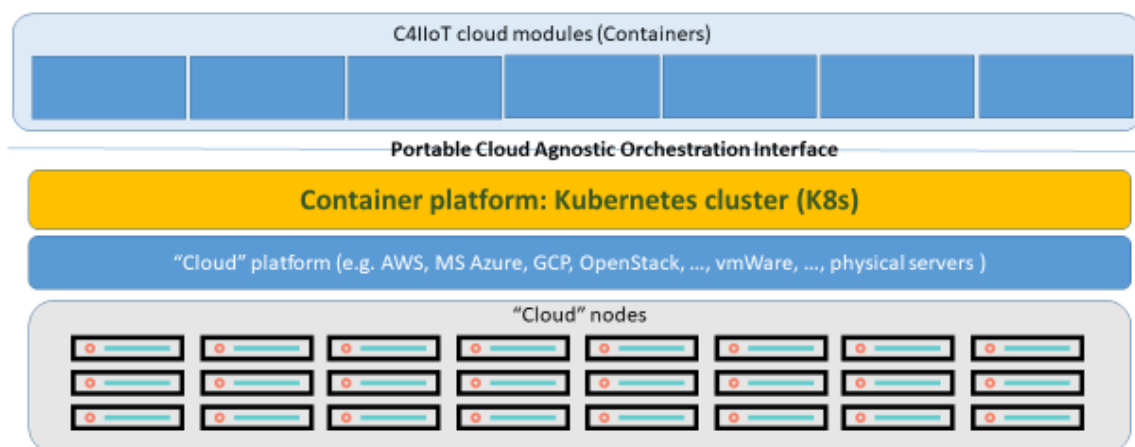
Thus, as shown in next picture, C4IIoT “dockerized” modules will be deployed through a portable cloud-agnostic orchestration interface to a cluster of Kubernetes nodes that can be hosted on any public, private or hybrid cloud, as well as on traditional virtualized or physical servers based Data Centers.

## Cloud Models Technologies



**Figure 13.** Cloud Models Technologies.

## C4IIoT modules Orchestrator



**Figure 14.** C4IIoT Modules Orchestrator.

### 3.2 Securing the Cloud Layer

Since the C4IIoT modules constitute a very sensitive part of the security mechanism implemented by the system, they also need to be protected to make sure the behaviour of the system is correct.

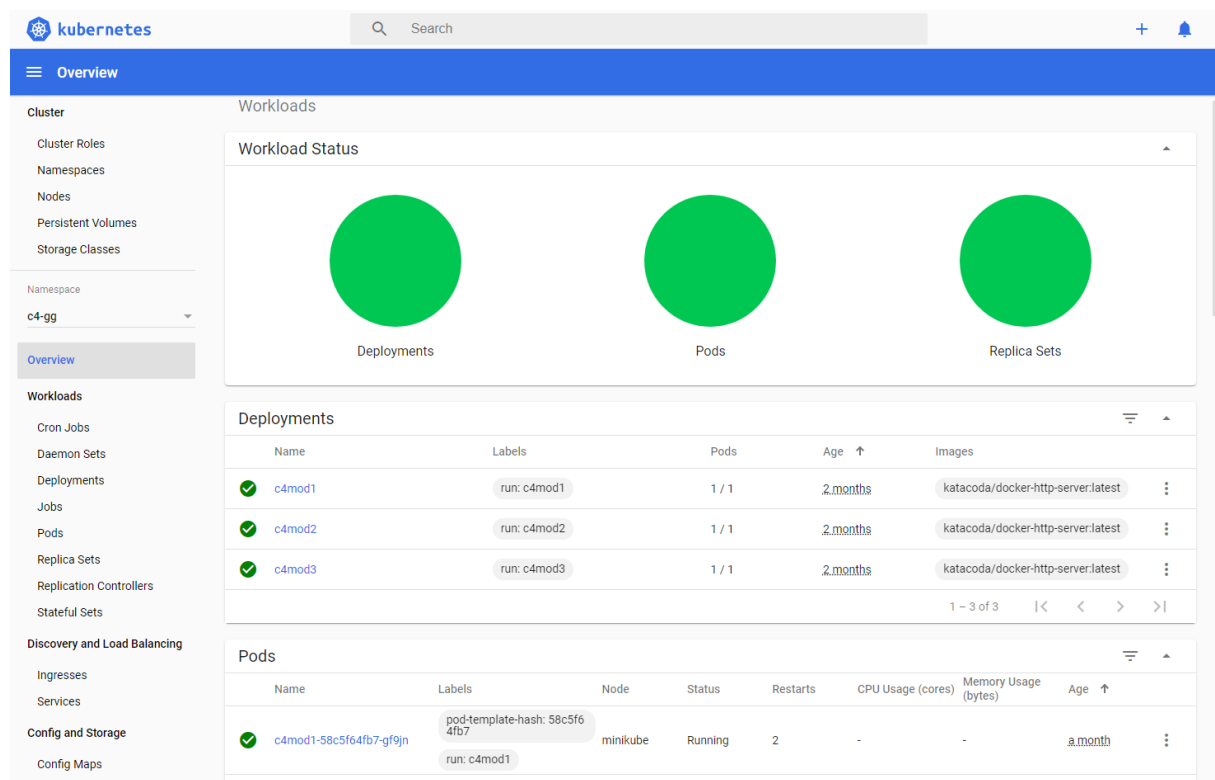
The following mechanisms are implemented inside the Cloud Layer Orchestration in order to secure the hosted C4IIoT modules

- C4IIoT dockerized modules communicate among each other only through predefined network ports with REST request/response APIs over HTTP/HTTPS (JSON or XML are both supported)
- Network isolation of the modules inside the Cloud Layer: all incoming network requests are managed through the Cloud Gateway
- Cloud Gateway enforces network traffic encryption: HTTPS for all external communications, client side authentication
- Cloud Gateway implements Web Application Firewall: OWASP ModSecurity rules are enforced on the managed traffic

All these features aimed at securing the Cloud Layer (and the overall C4IIoT system) are based on open-source components deployed inside the Kubernetes cluster.

The following sections describe the components mentioned above, that have been tested inside a small-scale Kubernetes (K8s) cluster based on MiniKube (<https://minikube.sigs.k8s.io/docs/>) running on a CentOS virtual machine inside a lab at HPE Italy HQ premises (Cernusco S/N, Milan, Italy).

The K8s web-based dashboard is used to get an overview of all the K8s objects like deployments, pods, services, ingress, replica sets, etc. The following screenshot of the MiniKube lab cluster provides only a glimpse of the Web-UI interface showing one K8s test namespace (Figure 15).

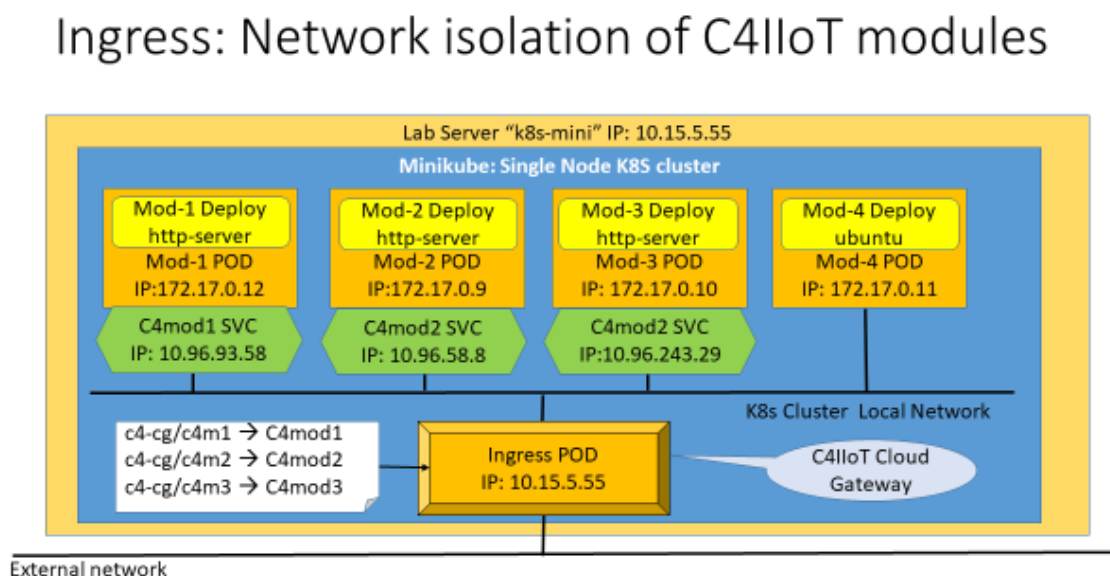


**Figure 15.** Kubernetes Web-UI interface showing one K8s test namespace.

### 3.2.1 Network isolation of C4IIoT modules: K8s Ingress is C4IIoT Cloud Gateway

C4IIoT modules running inside the Cloud Layer are connected among each other inside the K8s Cluster Local (internal) network that is not directly accessible from external nodes. The docker images and run-time resource requirements are described by K8s deployment objects, which are “executed” by the K8s pods (<https://kubernetes.io/docs/concepts/overview/components/> for an intro on K8s components). K8s services guarantee stable internal endpoints for the pod (executable container instances) independent from pod restart policies in case of failures.

The K8s Ingress component (<https://kubernetes.io/docs/concepts/services-networking/ingress/>) acts as a gateway between the external network and the internal K8s cluster network, and therefore is the perfect match to implement the C4IIoT Cloud Gateway. Network connections for C4IIoT modules exposing their services to other layers will be “published” on the Cloud Gateway (K8s ingress pod); external clients will send their REST requests to the Cloud Gateway address, that will dispatch (and load balance if necessary) to the internal module, based on rules (bottom left of Figure 16) parsing the URL components.



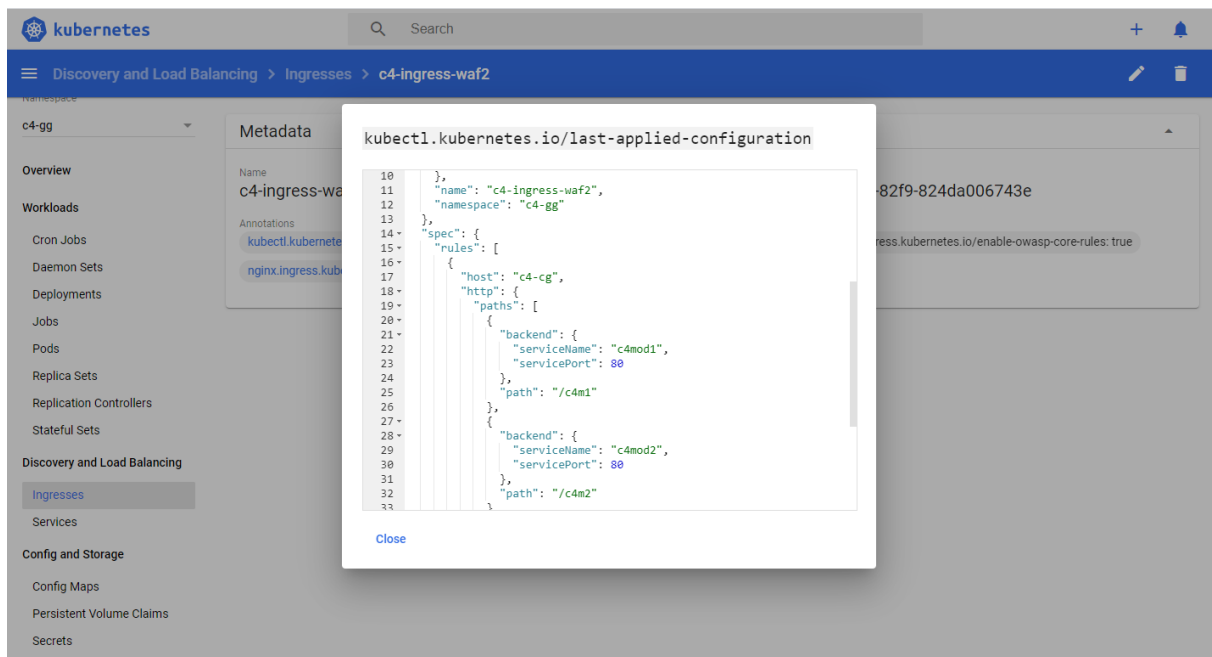
**Figure 16.** Network isolation of C4IIoT modules.

The Figure 17 highlights the K8s ingress configuration equivalent to the one presented in previous figure (snippet of the YAML config file displayed by the K8s Dashboard) showing the mapping rules of incoming URLs and internal (backend) services.

Besides being the single point of contact/entry endpoint, the Cloud Gateway plays a central role in the protection of the Cloud Layer components, aggregating and consistently applying the protection mechanisms described in the following sections to all incoming communications flows<sup>1</sup>.

<sup>1</sup> Also outgoing connections and intra-cluster local connections can be controlled using K8s Network Controllers, e.g. Calico, Istio, etc. These components are not compatible with the MiniKube configuration of the lab server, therefore they will be integrated in next releases when using a different cloud platform for the C4IIoT prototype integration on Kubernetes.



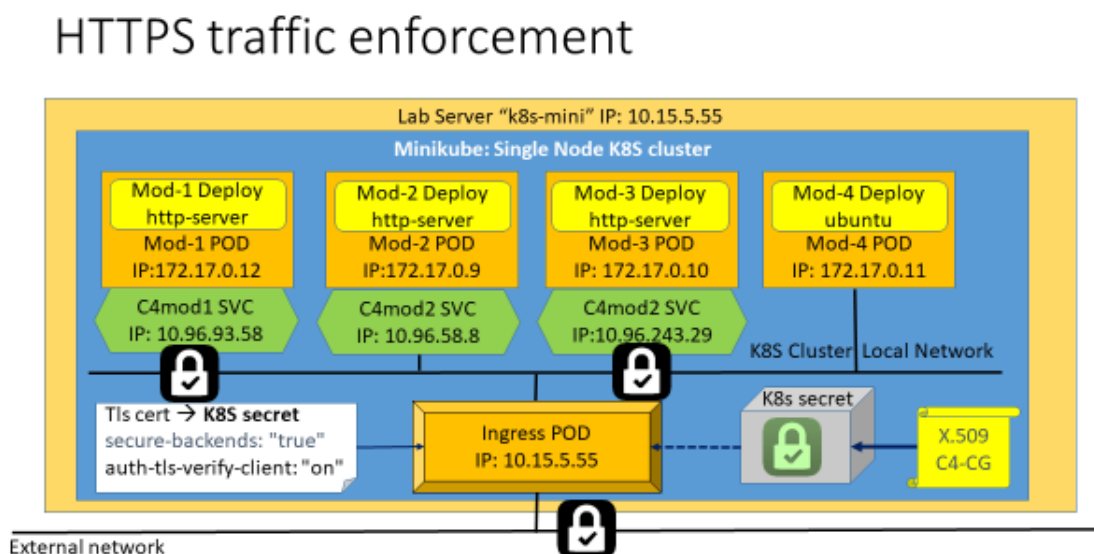


**Figure 17.** Mapping rules of incoming URLs and internal (backend) services.

### 3.2.2 Traffic encryption and authentication enforcement

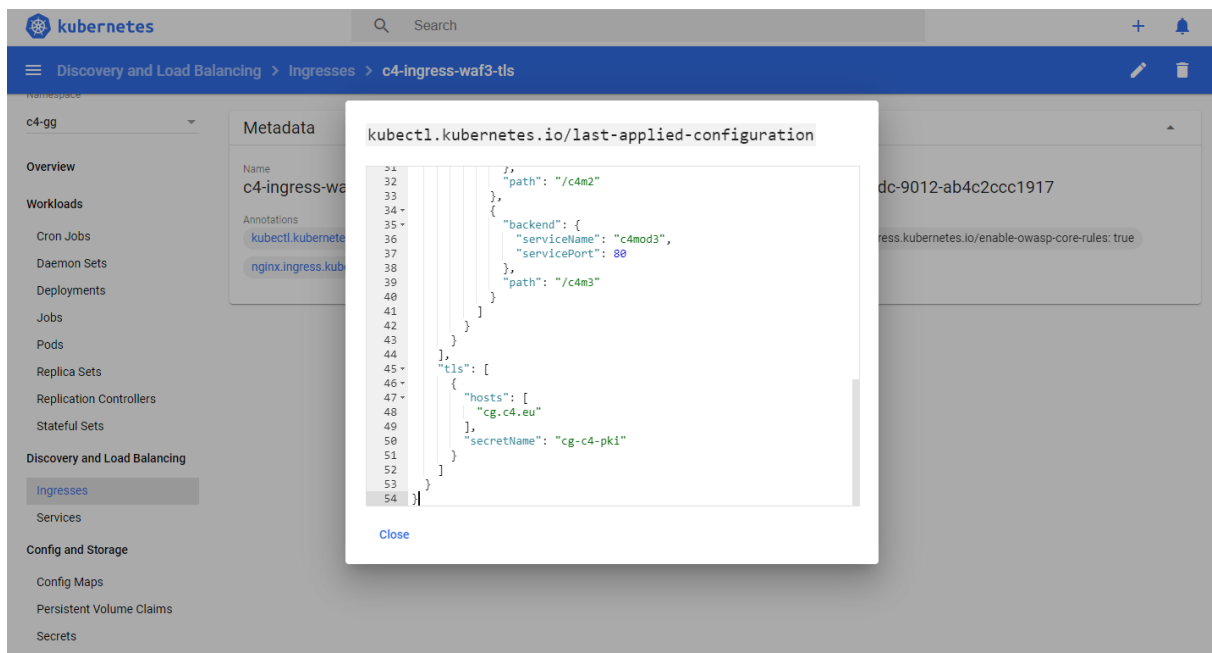
The Cloud Gateway (K8s ingress) configuration includes the capability to enforce privacy and authentication rules selectively by network route. Thus, for instance, it is possible to require that all incoming network connections need to use HTTPS (and automatically remap/upgrade each HTTP request to HTTPS).

The certificates and keys required by the TLS configuration do not need to be exposed inside the K8s configuration files and scripts. Instead they can be preventively stored inside K8s secret objects (saved in encrypted format), and then simply referred by the Cloud Gateway configuration, without exposing them to the view of system operators (depicted bottom right on the Figure 18).



**Figure 18.** HTTPS traffic enforcement.

The Figure 19 highlights the K8s ingress configuration equivalent to the one presented in previous figure focusing on the TLS setup using K8S secrets (snippet of the YAML config file displayed by the K8s Dashboard).



**Figure 19.** TLS setup using K8S secrets.

Moreover, also the internal network connection between Cloud Gateway and C4IIoT module can be configured as encrypted, regardless of the relative privacy of the intra-cluster local network.

Finally, the Cloud Gateway can impose access restrictions on the external clients, for instance by requesting TLS certificate validation for the client requests, as well as other compliance rules like for instance client IP white or black lists.

### 3.2.3 Cloud Gateway embedded Web Application Firewall

Traditional best practices for a component like the Cloud Gateway, as external endpoint for the Cloud Layer, impose the usage of a Firewall between the external network and the K8s ingress pod of the Cloud Gateway. And this will be definitely part of the full C4IIoT prototype deployment.

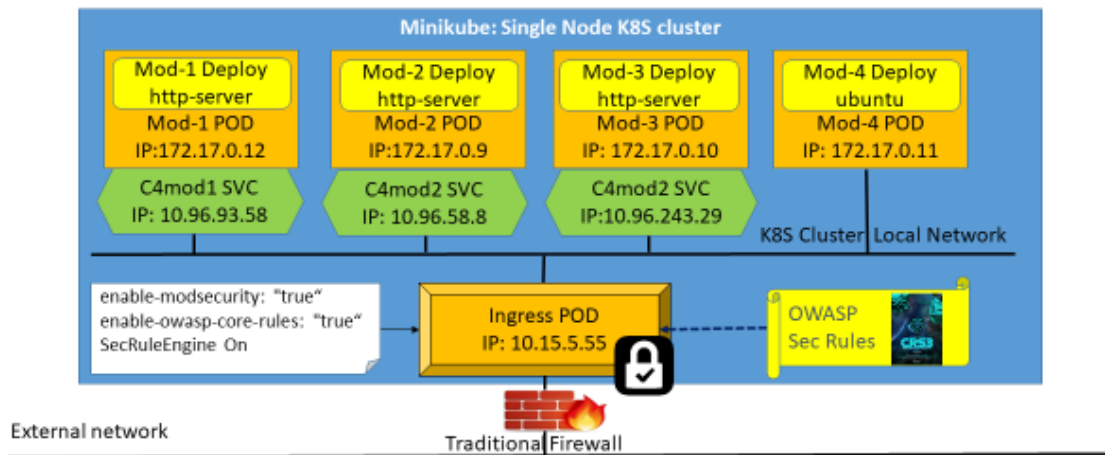
In addition, since the Cloud Gateway external interface is based on HTTPS REST request/response APIs, it might become an interesting target for attacks based on common hacker techniques developed for the Web sites.

OWASP Foundation (<https://owasp.org/>) has developed, and is actively updating, a set of attack detection rules to protect Web Services from a wide range of cyber-attacks: the OWASP ModSecurity Core Rules Set (<https://owasp.org/www-project-modsecurity-core-rule-set/>).

The Cloud Gateway is embedding an instance of ModSecurity engine, that will protect all C4IIoT modules inside the Cloud Layer from these common attacks, including for instance SQL Injection (SQLi), Cross Site Scripting (XSS), Local File Inclusion (LFI), Remote File Inclusion (RFI), Remote Code Execution (RCE), PHP Code Injection, HTTP Protocol

Violations, HTTPoxy, Shellshock, Session Fixation, Scanner Detection, Metadata/Error Leakages, Project Honey Pot Blacklist, GeoIP Country Blocking.

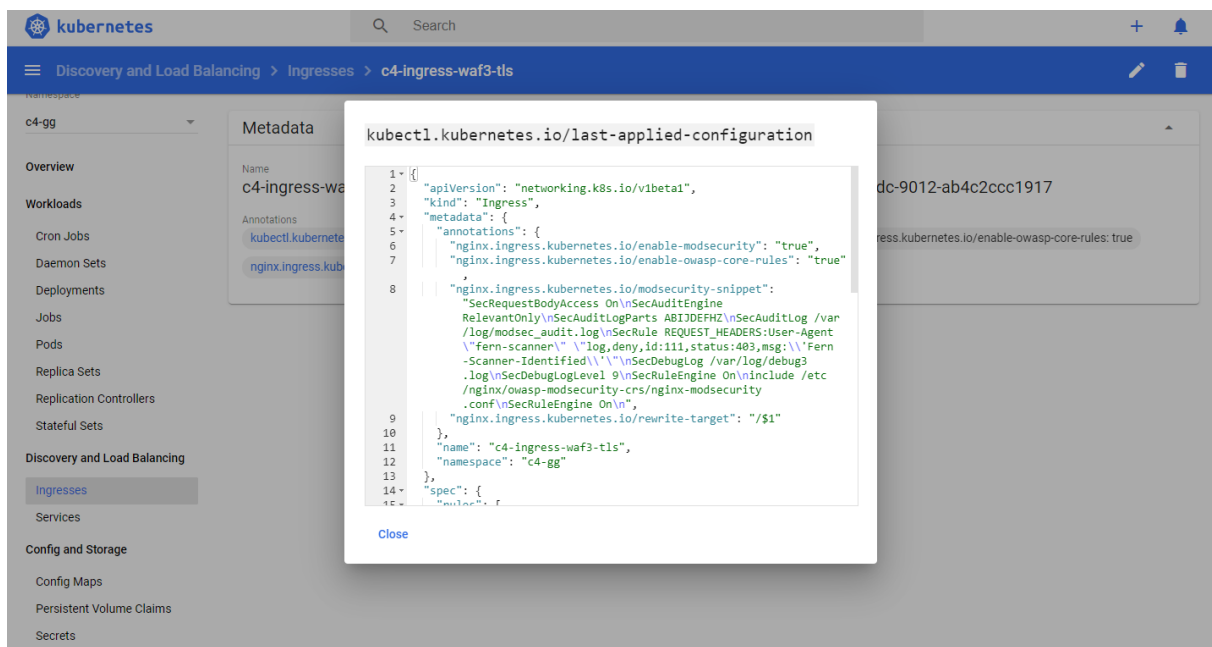
## Cloud Gateway Web Application Firewall



**Figure 20.** Cloud gateway web application firewall.

These rules are configured for the Ingress Pod and can be applied to all network connections managed by the Cloud Gateway, as depicted in Figure 20.

The Figure 21 highlights the K8s ingress configuration equivalent to the one presented in previous figure focusing on ModSecurity rules engine enablement (snippet of the YAML config file displayed by the K8s Dashboard).



**Figure 21.** ModSecurity rules engine enablement.

To be noticed, the OWASP Security Rules enforcement does not require any code change to the C4IIoT services that are automatically protected by the Cloud Gateway. Moreover, the general security best practices can be easily implemented by placing a traditional firewall between the external network and the Ingress POD endpoint.

### 3.3 Orchestration of Utility Services

In addition to the C4IIoT modules, the Cloud Layer will internally orchestrate some shared utilities supporting the overall security of other components and the system integrity:

- CA/PKI hosting for the management of certificates for all C4IIoT modules, both inside the Cloud Layer and for Level 2 and Level 1 modules/components
- Private Storage service for C4IIoT modules (ABE-protected data and others)
- Document repository system
- Docker image private repository with embedded:
  - Docker image signatures with Notary Service

#### *Docker image vulnerability scanning*

- Monitoring systems for cloud layer infrastructure and C4IIoT components

All these services are based on open-source components deployed inside the Kubernetes cluster.

#### 3.3.1 Certificate Authority services

The Cloud Gateway will enforce the usage of HTTPS connection and therefore the usage of X.509 certificates for the different client and server modules. Moreover also device sensors will need certificates to be authenticated, and finally the Attribute Based Encryption mechanism (part of WP3 Task 3.4) will need certificates with specific roles.

In the context of WP4, as described in D4.1 chapter 2 “Identity Management”, an open-source Certificate Authority (CA) and Public Key Infrastructure (PKI) has been identified for the specific needs of the C4IIoT project: EJBICA <https://www.ejbica.org/>

**EJBICA**  
PKI by PrimeKey

## Administration

Version : EJBICA 6.15.2.6 Community (r34564)

**Welcome to EJBICA Administration.**

Node hostname : ejbica-deployment-6c6b4fc7db-q284w  
Server time : 2020-04-20 14:03:51+00:00

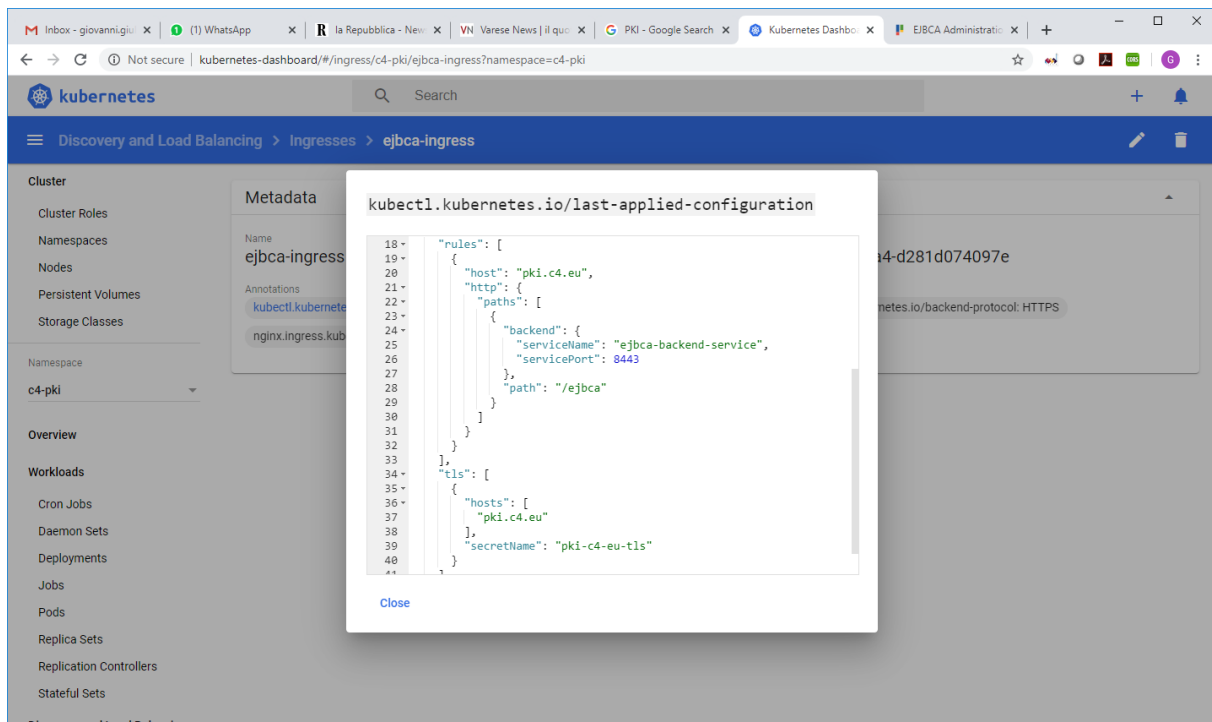
CA Status [2]			Publisher Queue Status [2]	
CA Name	CA Service	CRL Status	Publisher	Length
ManagementCA	OK	OK	No publishers defined.	

© 2002–2020 PrimeKey Solutions AB. EJBICA® is a registered trademark of PrimeKey Solutions AB.

**Figure 22.** EJBICA main administration Web-GUI.

The structure of the CA usage in the context of the project is described in D4.1, while here the deployment options are presented. EJBCA Community version is available as dockerized containers in “test” mode (single container) and “production” mode (with load-balanced front-ends and independent DB containers). The Figure 21 shows the main administration Web-GUI.

Both mode versions have been tested and K8s specific YAML configuration files have been setup to automate the deployment on the MiniKube cluster. In addition a specific set of ingress rules have been created to support external access to both the Web interface of EJBCA and the REST protocols for certificate updates. The Figure 23 highlights the K8s ingress configuration focusing on EJBCA service backend port remapping and TLS support (snippet of the YAML config file displayed by the K8s Dashboard).



**Figure 23.** EJBCA service backend port remapping and TLS support.

For the purpose of MVP support, only the CA/PKI is requested, therefore – in absence of the full orchestrator system – it will be deployed as docker containers in “production” mode (with high availability features enabled).

### 3.3.2 Private Attribute Based Encryption Protected Storage

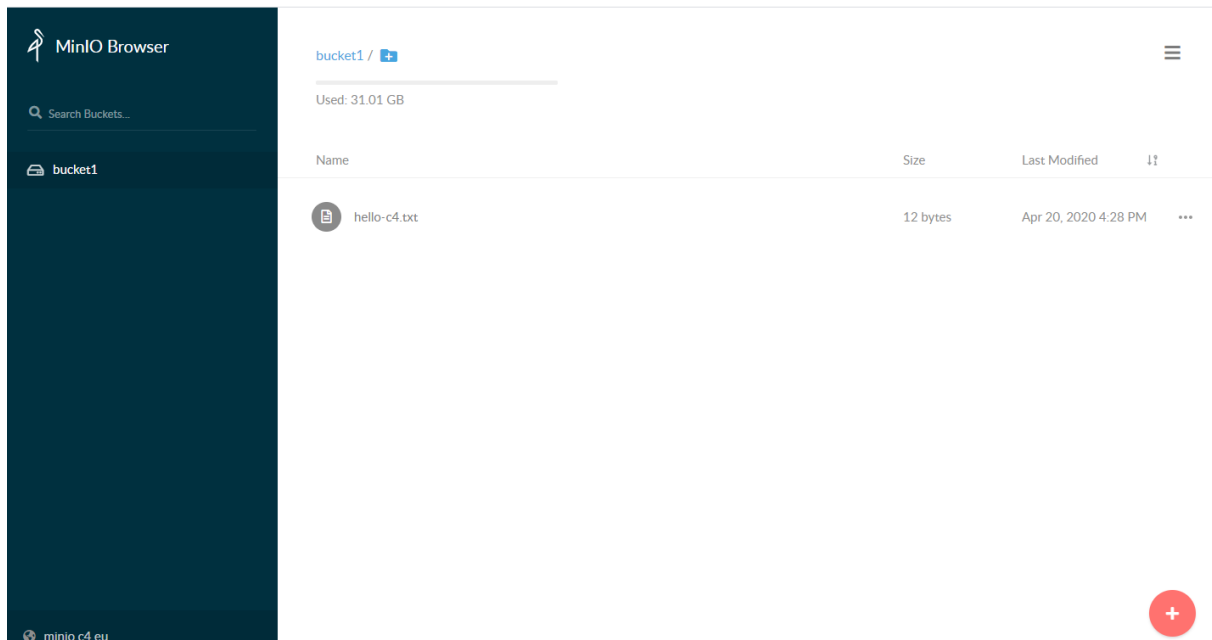
The previously mentioned Attribute Based Encryption mechanism (ABE, part of WP3 Task 3.4) uses BlockChain technology to create a system to be used for sensible data produced inside C4IIoT, allowing a fine granularity access control to these data. The data managed by ABE is not stored inside the BlockChain itself, which contains the authorization records for the data and a URL to the location where the encrypted copy is stored.

Thus, a private storage system has been identified to support ABE mechanism. This storage system will be private to C4IIoT and hosted inside the Cloud Layer, and accessible to other layers as well. The open-source Object Storage system that has been selected is MinIo <https://min.io/>

MinIo implements an AWS S3 compliant Object Storage system, with buckets and data objects; it has a powerful standard REST API, as well as a large variety of SDK for various

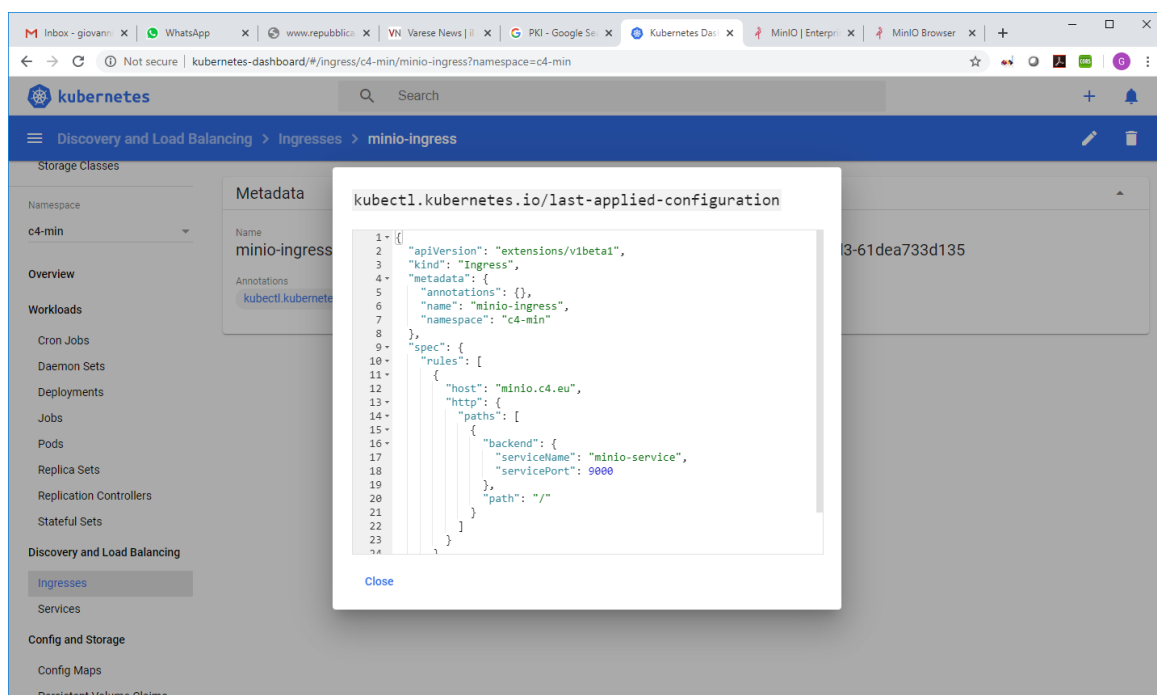
programming languages (Java, Python, GoLang, JS, etc.) – this will facilitate the integration inside C4IIoT modules written in different languages.

A set of command line interfaces is also available, e.g. the native “mc”, “awl-cli”, “s3cmd”, etc.). The Web-GUI interface is used for management and – having the correct credentials - for browsing the stored data, as shown in Figure 24.



**Figure 24.** MinIo web GUI.

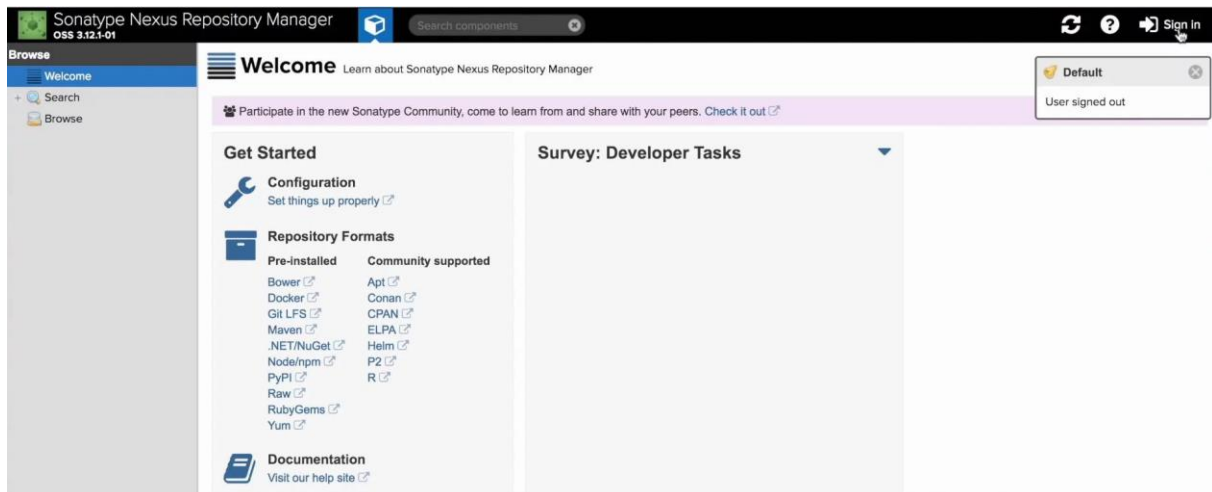
MinIo is also “K8s-friendly”, offering a set of YAML configuration files to deploy the Object Store inside a K8s cluster. Since the service needs to be reachable also outside the Cloud Layer, a set of ingress rules has been setup for port redirection to this service. Figure 25 highlights the K8s ingress configuration focusing MinIO backend port remapping (snippet of the YAML config file displayed by the K8s Dashboard).



**Figure 25.** K8s ingress configuration focusing MinIO backend port remapping.

### 3.3.3 Private Artifact Repository

A private artifact repository protected inside the Cloud Layer can be considered a “nice-to-have” feature, but its importance can raise if, besides storing generic artifacts (e.g. Maven/Java, npm, NuGet, Helm, P2, OBR, APT, GO, R, Conan, etc. components) in a safe and privacy protected way, it can serve as private docker image registry as well. This is the main reason why the open source version of Nexus 3 ( <https://www.sonatype.com/nexus-repository-oss> ) has been tested inside the C4IIoT K8s lab. Figure 26 shows the main page of the Web admin GUI for Nexus 3 open source version.



**Figure 26.** The main page of the Web admin GUI for Nexus 3.

Nexus3 has been deployed inside the K8s cluster and successfully tested as Docker Image registry by pushing and pulling images to it, both directly from docker command line as well as using K8s configuration YAML files (k8s deployment objects referring to this repository).

Then the Docker Notary service ( <https://docs.docker.com/notary/> ) has also been setup, and signing images tests have been performed using the official instructions about Docker Content Trust (DCT) at [https://docs.docker.com/engine/security/trust/content\\_trust/](https://docs.docker.com/engine/security/trust/content_trust/)

The Client Side enforcement of image signing at Docker push time and image signature check at Docker pull time using DOCKER\_CONTENT\_TRUST and DOCKER\_CONTENT\_TRUST\_SERVER environment variable works fine with Nexus3 repository. Unfortunately, this Client Side mechanism, designed for Docker CLI and API, is not compatible with Kubernetes, that is not using such environment variables.

The Run Time enforcement of image signature checks is supported only in Docker Enterprise Engine, and thus not available for the Community edition.

For these reasons, i.e. the obstacles in DCT for a K8s and the requirement for a paid version of docker engine, the usage of Nexus3 as private docker registry with an external Notary was not successful, although the Nexus3 repository may still be used inside the project for other artifacts.

### 3.3.4 Secure Private Docker image registry

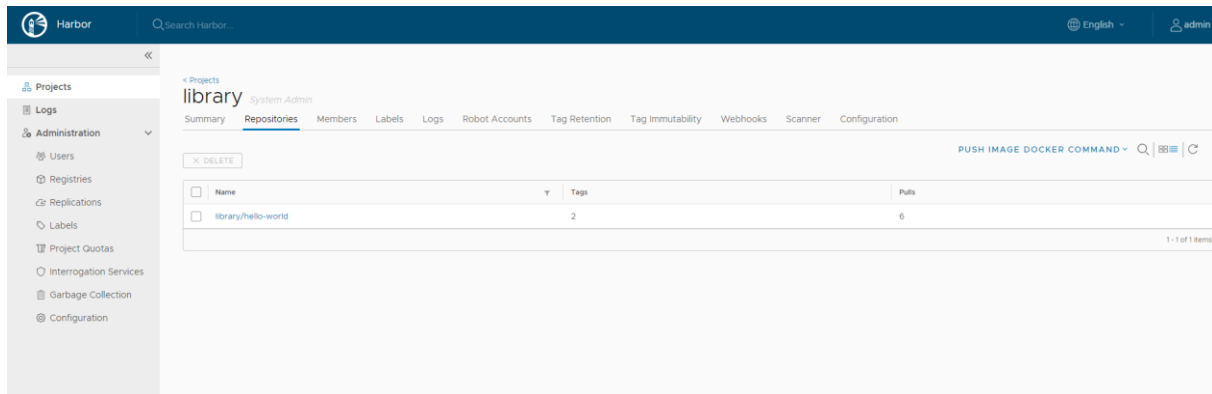
The advantages in terms of security and flexibility of a private docker image registry are obvious, and Nexus3 had been identified exactly for this reason, but the issues related to its integration with DCT and K8s suggested to look for an alternative solution that natively integrates docker images signing.



The open source Harbor project ( <https://goharbor.io/> ) is offering a trusted cloud native repository for Kubernetes that integrates all the features needed by C4IIoT:

- Private docker image registry
- Integrated control on image signing process
- Integrated control on image vulnerabilities

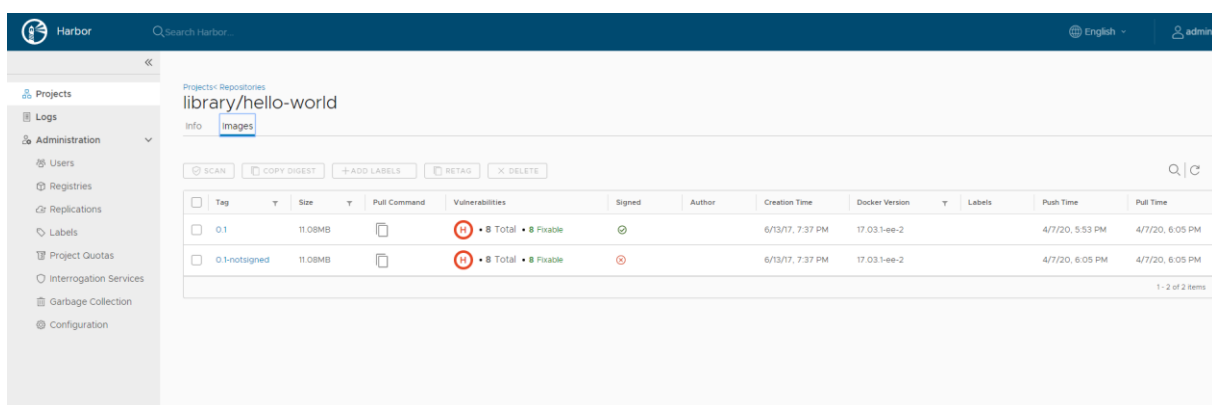
Figure 27 provides an overview of a sample image set for “hello-world” (2 tags) pushed on the MiniKube lab deployment of the Harbor system.



**Figure 27.** An overview of a sample image set for “hello-world” (2 tags) pushed on the MiniKube lab deployment of the Harbor system.

With Harbor docker registry the Client Side DCT process can be used to sign images at Docker push time. The DCT Notary is integrated inside Harbor: it is deployed with Harbor containers and internally used by Harbor registry to check image signatures - if required by the server registry configuration – when clients issue a Docker pull ( <https://goharbor.io/docs/1.10/working-with-projects/project-configuration/implementing-content-trust/> ). This mechanism is independent from any Docker client environment variable setting and works also on the community edition of the docker server.

Figure 28 shows the expanded view of previous image, where the two versions of the “hello-world” image differ in the fact that the first one is signed, while the second one has been pushed without a signature process.

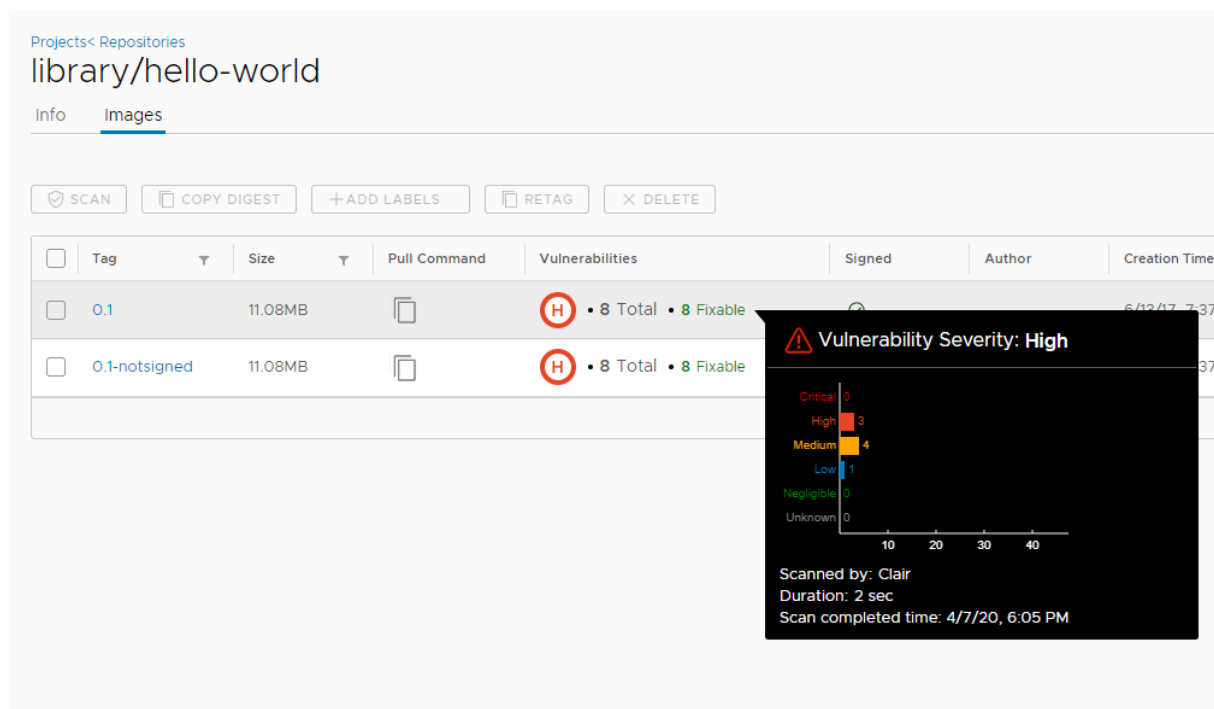


**Figure 28.** Differences between the two versions of the “hello-world” image.

In addition, Harbor also integrates Clair docker image vulnerability scanning tool ( <https://coreos.com/clair/docs/latest/> ). When images are pushed to the registry, a Clair scan is performed and the image is “rated” based on the found vulnerabilities ( <https://goharbor.io/docs/1.10/administration/vulnerability-scanning/> ). In the previous

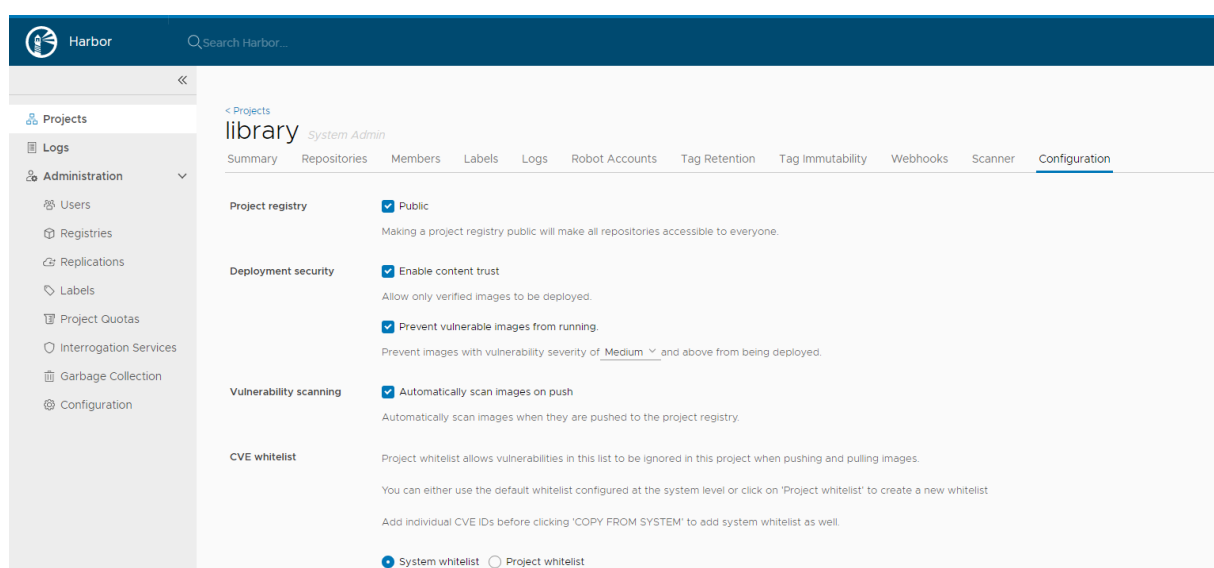


screenshot, both “hello-world” images have the same contents (they differ only from the presence signature), and the summary of the scan (vulnerability score) can be displayed, as shown in Figure 29.



**Figure 29.** The summary of the scan (vulnerability score).

The most interesting characteristics of Harbor from C4IIoT perspective is that both image signature verification and vulnerability score can be configured selectively on a per project (registry section) basis. In \, for the previous sample project, the Deployment security sections of the configuration states “allow only verified images to be deployed” (i.e. signed images) and “prevent images with vulnerability severity of Medium and above to be deployed” (images with medium-high score).

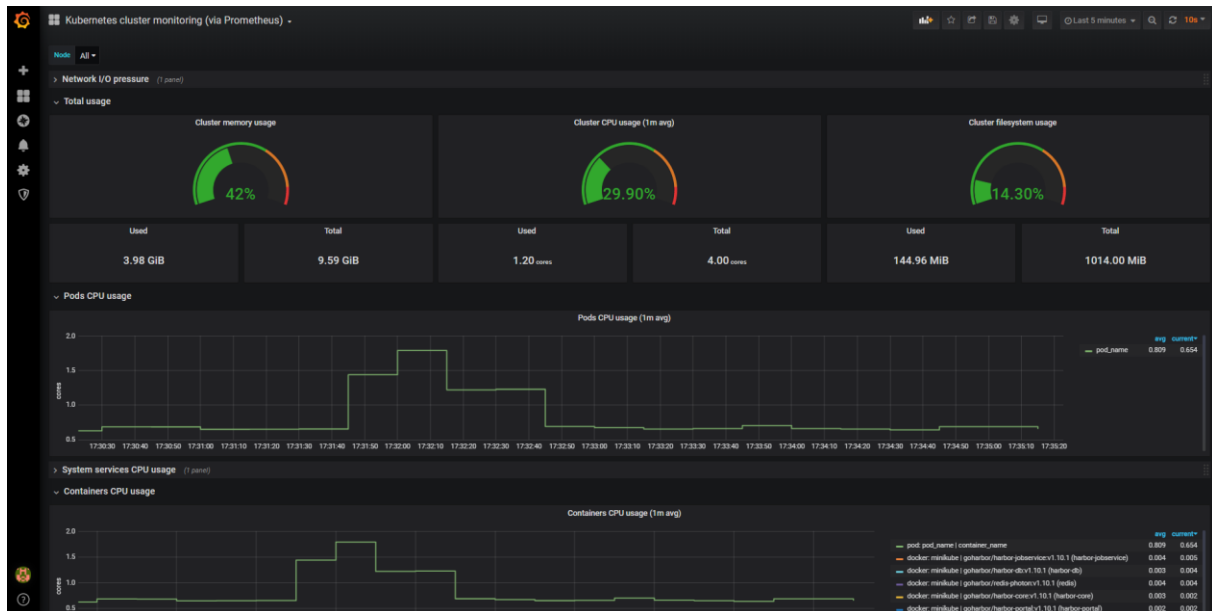


**Figure 30.** The Deployment security sections of the configuration.

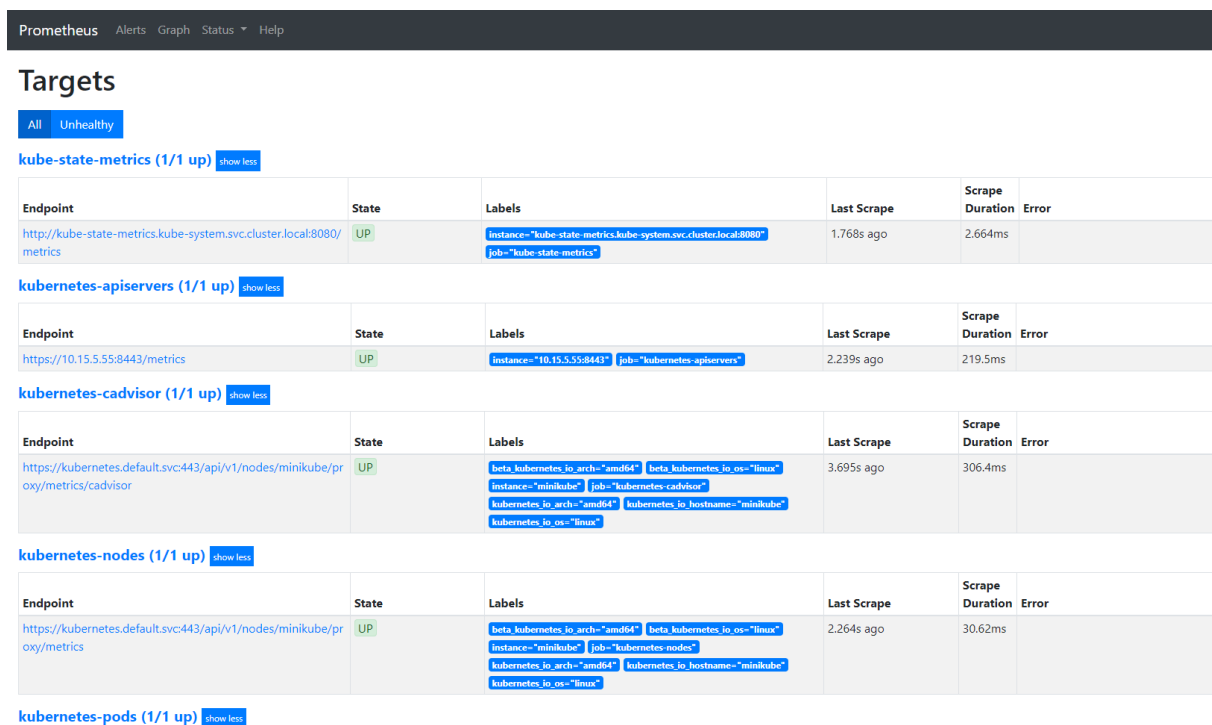
All this configuration is performed at Harbor server side, independently from the client setting, thus it is automatically enforced for simple docker clients, on community version of docker engine as well as on Kubernetes.

### 3.3.5 Infrastructure and component monitoring

To manage a complex system like C4IIoT, a monitoring system is required to get a clear and consistent view of the system status, as well as resource utilization of the components/modules and the overall infrastructure.



**Figure 31.** Kubernetes cluster monitoring dashboard deployed on the Minikube cluster for C4IIoT.



**Figure 32.** A sample view of the state of Prometheus targets on the MiniKube cluster for C4IIoT lab.

Grafana (<https://grafana.com/oss/grafana/>) is an open source very flexible and powerful system to create impressive graphical dashboards; it can be connected to a large variety of data sources of different kinds. Prometheus (<https://prometheus.io/>) is the specific open source data source for docker. Figure 31 shows a Kubernetes cluster monitoring dashboard deployed on the Minikube cluster for C4IIoT: the first row has gauges for Cluster memory usage, Cluster CPU usage and File system usage; the next rows show other graphs by POD and Container.

These data are monitored by Prometheus data source and passed to Grafana for dashboard real time rendering. Prometheus pulls information out of its “targets”, i.e. docker and k8s infrastructure components. Figure 32 shows a sample view of the state of these targets on the MiniKube cluster for C4IIoT lab.

The Grafana dashboards can be easily customized and extended to other metrics that will be identified in the future for C4IIoT components.

## 4. Raising security and privacy-awareness using Intel SGX

Today, organizations and enterprises tend to outsource data processing workloads to cloud providers. Providing applications as-a-service has become a very convenient trend due to lower cost and maintenance complexity. Still, these workloads contain important information about the user or the organization. Processing sensitive information (such as user files, e-mails, logs, network traffic, etc.) needs to be taken seriously by complying to security and privacy preserving standards in order to guarantee confidentiality. Yet, it is quite challenging to provide strong guarantees regarding the safety of users' data, especially when handled by parties other than themselves [5,6].

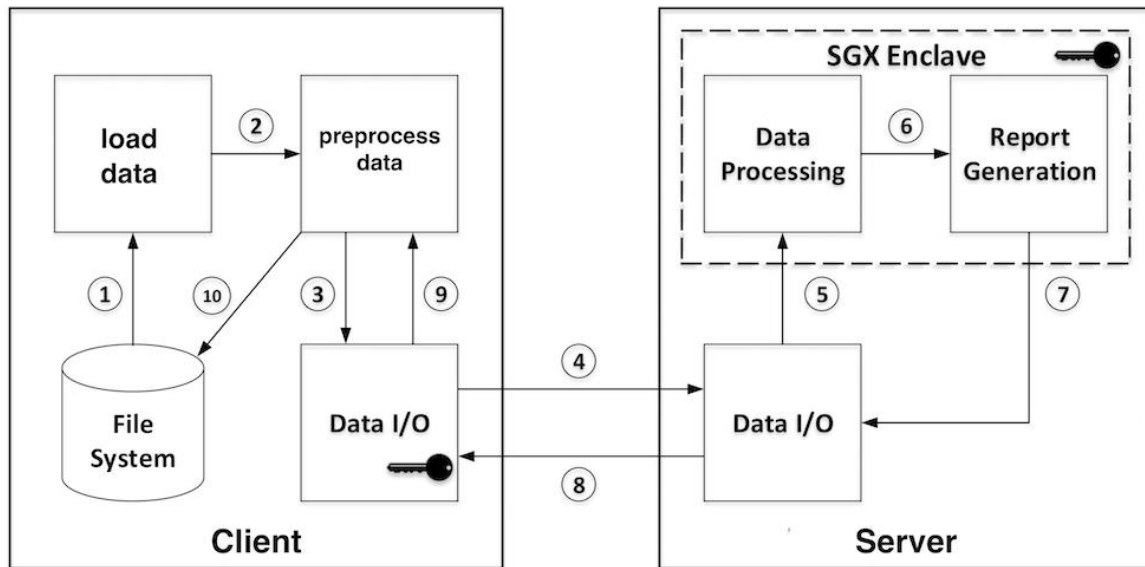
### 4.1 Intel SGX Essentials

Intel SGX is a hardware assisted mechanism in the form of an ISA extension to the Intel architecture. It is designed to allow secure attestation and sealing to application software executing in a secure environment that is known as enclave. The main purpose of these extensions is the protection of selected code parts and data from disclosure or modification in untrusted environments. The enclaves are protected by the CPU that is in charge of any access to the enclave memory or other protected areas of execution. Any instruction that reads or writes to the enclave and is not part of it, fails. Assuming an untrusted or even a malicious operating system, hypervisor or firmware, SGX protects the confidentiality of the enclave pages. An Intel SGX application consists of (i) the trusted code and (ii) a trusted enclave that it securely calls into.

The code and data that are part of the enclave are stored in a DRAM subset, the Processor Reserved Memory (PRM). PRM has a contiguous range and cannot be accessed by any system software or peripherals. Moreover, the contents of the enclaves are stored into the Enclave Page Cache (EPC), a subset of PRM. Non-enclave software is not able to access the EPC. For Intel Skylake CPUs, the EPC size is between 64 MB and 128 MB and SGX provides a paging mechanism for swapping pages between the EPC and untrusted DRAM. The data of the enclave that has to be written to the disk is encrypted and checked for its integrity. Between enclaves, SGX enables local attestation. Additionally, in the case of a third-party application or software, SGX allows remote attestation to ensure that the application is uncompromised, and therefore can be trusted. SGX enables the remote system to establish a connection with the enclave, using an end-to-end encrypted channel. Using remote attestation, the client challenges the server to verify that the core part of the engine is located inside a signed SGX enclave, executed on an SGX-enabled processor. In this way, we eliminate the possibility of a malicious entity posing as an SGX-enabled server in order to obtain access to a user's private data.

### 4.2 Methodology

Our design is depicted in Figure 33. We define three different entities: (i) the client, which transmits the necessary data to the remote server for processing, (ii) the server, which is responsible for performing the analysis in a privacy-preserving way, and (iii) the public cloud provider. The entire processing is performed in the cloud-based server, encapsulated inside Intel SGX enclaves, which communicates with the clients through a network connection. This encapsulation enables the protection of the data processing algorithms, and most importantly the privacy of the user's data. We assume that a client is installed and initially executed when the device is in a clean state, so no malicious executable has taken the control of the client or the device.



**Figure 33.** Our approach for privacy-preserving data processing in third-party clouds.

A public cloud environment is typically considered untrusted, since there is no control over the operating system, the hypervisor, the drivers, the management stack, the system's memory, I/O devices, etc. Furthermore, even in a fully healthy environment, there is always the possibility of an honest-but-curious cloud provider, willing to learn and extract information regarding the users or the system utilization. In C4IIoT, we safeguard both end-users and the server from the aforementioned conditions. Obviously, the client and the server are required to safeguard the transmission and the processing of the user's data. For the purpose of completeness, we assume an uncompromised Intel SGX enabled processor hosting the remote server. Finally, we stress that handling any side-channel attacks against Intel SGX is out of our scope.

## 5. HPE Introspect tool replacement

Based on our discussion during the C4IIOT kick-off meeting and since that the HPE Aruba introspect tool was not available for this project and based on the fact the product is not anymore available (product in EOL from 01/05/2019 and end of support on 01/05/2020 [7]), we pursuit to identify the behavior-based attack and allow the incident investigation and response at enterprise scale. The solution allowed to identify the attacks involving malicious, compromised or negligent users. Systems and devices are found and remediated before they damage the operations and reputation of the organization.

In the market, there are several commercial solutions for EUBA, which could help to replace the UEBA solution. In C4IIOT, we would like to investigate open source UEBA solutions. Here, we identify a couple of solutions:

- The ELK Stack [8], is a collection of three open-source products: Elasticsearch, Logstash, and Kibana. These three tools can be used for visualization and analysis of IT events.
  - Elasticsearch is a NoSQL database based on the Lucene search engine.
  - Logstash is a log pipeline tool that accepts input from various sources, performs different transformations and exports data to various targets.
  - Kibana is a visualization dashboard that works on Elasticsearch.
  - The stack also includes a family of log shippers or log forwarders called Beats, which together with the other 3 products have made the "Stack ELK" a lot of success in recent times.
- The OSSIM SIEM [9] platform, includes event collection, normalization, and correlation. Open Source Security Information and Event Management (SIEM), provides a feature-rich open source SIEM, completed with event collection, normalization and correlation. Launched by security engineers, OSSIM was created specifically to address the reality many security professionals face: A SIEM, whether it is open source or commercial, is virtually useless without the basic security controls necessary for security visibility.

With these products we can elaborate and create correlation rules that allow us to have the UEBA functionalities.

## 6. Concluding Remarks and Next Steps

The deliverable provides detailed description of Behavioural Analysis and Cognitive Security Framework, with detailed specification of its packages, demonstration of its usage and with results obtained in the process of testing using datasets provided by the project partners. In the continuation, we gave description of solutions for building, deploying, orchestrating and managing heterogeneous hybrid cloud environments including tools like Kubernetes, MinIO, Grafana, Prometheus, etc. In the third part we deliver the text describing details of Intel SGX technology and its possible usages within the C4IIoT system. Finally, options to replace the HPE Introspect tool have been described.

Within the MVP deliverable that is due at the same time as this deliverable, BACS modules developed so far have been successfully integrated into the C4IIoT framework providing an important component for the minimum viable product demonstration.

In future steps, implementation of BACSCL, package providing anomaly detection based on deep learning with data and model partitioning in cloud environments is to be implemented. The package will include privacy aware techniques in data analysis. The implementation is in its initial phases.

Upon finishing of this component, we plan to execute a thorough evaluation of both unsupervised and supervised BACS models on labeled datasets from C4IIoT simulated environments. It is also worth to note that even unsupervised AD models cannot be fully verified without labeled data. Acquisition of labeled datasets within the C4IIoT project represents one of the challenges that will be tackled in the immediate future.

All C4IIoT data provided for the purpose of testing of BACS components developed so far is unlabeled and we could use it only to train unsupervised AD models.

The final step, running somewhat in parallel with the other described tasks includes setting up the system that will allow running of certain behavioural models on using Intel SGX instructions within the software environments provided by FORTH for those purposes.

The cloud layer of the C4IIoT architecture is hosting, managing and orchestrating execution of several modules like Mitigation Engine, Security Assurance, ML-based Behavioural Analysis (BACSCL), Advanced Anomaly detection, etc.

This deliverable gives a detailed description of our approach to provisioning and configuring the infrastructure resources needed by the cloud-hosted C4IIoT modules, creating automated mechanism to seamlessly integrate and configure cloud resources and supporting building, deploying and managing C4IIoT modules inside a hybrid cloud environment.

## 7. References

1. <https://docs.aws.amazon.com/>
2. <https://cloud.google.com>
3. <https://azure.microsoft.com>
4. <https://www.openstack.org>
5. D. Deyannis, E. Papadogiannaki, G. Kalivianakis, G. Vasiliadis, S. Ioannidis. TrustAV: Practical and Privacy Preserving Malware Analysis in the Cloud. In Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy (CODASPY). March 2020.
6. Victor Costan and Srinivas Devadas. Intel SGX explained. Technical Report. Cryptology ePrint Archive, Report 2016/086, 2016.
7. <https://www.arubanetworks.com/support-services/end-of-life/#IntroSpectSoftware>
8. [www.elastic.co/elk](http://www.elastic.co/elk)
9. <https://cybersecurity.att.com/products/ossim>



## Appendix - Experimental analysis of unsupervised BACS AD models in the smart factory use case

