Hardware-Performance-Counters-based anomaly detection in massively deployed smart industrial devices

Malcolm Bourdon
EDF R&D, LAAS-CNRS
Palaiseau, France

Pierre-François Gimenez and Eric Alata and
Mohamed Kaaniche and Vincent Migliore and
Vincent Nicomette
LAAS-CNRS, Toulouse, France

Youssef Laarouchi
EDF R&D, Palaiseau, France

***Abstract*—Energy providers are massively deploying devices to manage distributed resources or equipment. These devices are used for example to manage the energy of smart factories efficiently or to monitor the infrastructure of smart-grids. By design, they typically exhibit homogeneous behavior, with similar software and hardware architecture. Unfortunately, these devices are also of interest to attackers aiming to develop botnets or compromise companies' security. This paper presents a new protection approach based on Hardware Performance Counters (HPC) to detect anomalies in massively deployed devices. These HPC are processed using outlier detection algorithms. Compared to existing solutions, we propose a lightweight approach based on a comparative analysis of devices' HPC without relying on the modeling of the software applications running on the devices. To assess the relevance and the effectiveness of the approach, a thorough experimental analysis is carried out in a representative industrial-type environment, sampling the data from 100 Raspberry Pi to simulate about 10,000 devices deployed simultaneously. The results show high detection and performance efficiency under different software profiles and attack payloads. Moreover, the calibration of the approach depends primarily on the hardware rather than the application software running on the devices. It should ease its deployment in an operational environment.**

***Keywords*—Cybersecurity, Outlier detection, Hardware performance counters, Internet of Things (IoT).**

I. Introduction

Today, the deployment of the Internet of Things (IoT) technologies is growing in all application areas. These technologies aim to improve quality of life by providing *smart* environments. In this kind of environment, the different actors work together to make better or automated decisions, e.g., to optimize energy consumption, improve home protection against physical intrusion, etc.

Unfortunately, the massive and fast deployment of connected devices raises major security concerns. Many of these devices are manufactured without seriously tackling security threats. As a consequence, several vulnerabilities have been disclosed in commonly used devices that have become an easy target for attackers [1]. For example, the Mirai attack [2] took advantage of default settings and poor telnet implementation in several smart devices to compromise and use them in massive attacks such as DDoS.

Several research studies are being carried out to develop efficient security protection solutions adapted to IoT constraints and requirements [3]. In this paper, we will focus on anomaly detection in industrial contexts where IoT devices with similar configurations are massively deployed, connected to their environment by sensors and actuators and remotely managed by a service provider via a central server. Such devices usually include inexpensive hardware offering limited resources and implement simple functions with a quite stable behavior in an industrial context, which however may differ from one deployment to another (due e.g., to user-defined options or updates). A typical use case is the massive deployment of devices by energy providers to optimize energy consumption on a large scale. The connected devices can then be used to efficiently manage the power grid or to manage homes or buildings self-consumption.

The corruption of massively deployed connected devices by a malware could have negative consequences on the service provider in terms of delivered service and brand image.

Different strategies have been considered for detecting attacks targeting connected devices [4]. A major challenge is the limited resources of these devices (energy, CPU, memory, etc.), making the implementation of many common security approaches inadequate.

An approach to monitor such devices consists in modeling the legitimate behavior of the application running on it and detecting at runtime significant deviations from this model. One major limitation of this approach is the need to rebuild this model each time the application is updated, which may be difficult and costly for the service provider.

The approach described in this paper does not rely on the modeling of the legitimate application software. Instead we propose a lightweight mechanism to remotely monitor and compare the behavior of these massively deployed devices in order to identify outliers. The main assumption is that the majority of devices are legitimate and only a few of them exhibit "abnormal" behavior. To do so, we rely on the analysis of low level information to detect the anomalous behavior

978-1-7281-8326-8/20/$31.00 ©2020 IEEE

of devices. As we envision a scenario with several thousand of devices, our approach involves minimal additional network traffic. Also, the additional monitoring software installed on the devices for intrusion detection must deal with the limited memory and computing resources of the devices. The low level information investigated in our study correspond to *Hardware Performance Counters* (HPC) stored in the devices' processor. These counters record the occurrences of "low-level" events. They are available on a large number of different processors, which means that our solution can be adapted to a large number of hardware architectures.

To summarize, the contributions of this paper are the following:

- We develop a lightweight method to detect compromised devices among a massive population of similar devices. To do so we analyze then compare the processor's *Hardware Performance Counters* using existing outlier detection algorithms. This method incurs low computational and network traffic overhead, without modeling the behavior of the applications running on the devices. It can also be easily calibrated and adapted to different execution platforms.
- We perform thorough experiments based on 100 connected devices with various software profiles, attack payloads and outlier detection algorithms, for the practical validation of our approach. These experiments were calibrated and configured with the knowledge gathered from real energy provider use-cases. We are not aware of similar experiments to assess the relevance of HPC for detecting compromised IoT devices.
- The results obtained show that a high detection efficiency can be achieved, with a low overhead and execution time. Three outlier detection algorithms among the eight commonly used that we evaluated provide comparable results, significantly outperforming the other algorithms.
- We will make our code, data and results publicly available to facilitate the reproducibility of our work and to enable the development of other anomaly detection approaches for IoT devices based on HPC.

The paper is organized as follows. Section II presents the context, assumptions and the threat model. Section III discusses related research works and their limitations considering our assumptions and threat model. Section IV describes our anomaly detection approach along with the different steps that are executed either on the devices or the central server. Section V presents the set of experiments we carried out to assess the relevance of the approach and highlights the main lessons learned. Finally, Section VI concludes the paper and outlines future work.

II. Target environment and threat model

We describe first the assumptions about the environment, the devices and the network communications considered in our study, then we discuss the threat model.

A. Assumptions

In this paper, we consider the context of a massive deployment of connected devices disseminated through a large geographic area. Such context corresponds for example to a typical operational environment deployed by an energy service provider. The devices communicate with a central server to report relevant information. We assume that all the devices implement the same hardware architecture with a lightweight operating system and an application software that implements simple functions. These simple functions are assumed to exhibit a stable periodic behavior on average during predefined time periods, that is comparable from a device to another.

Moreover, as some software updates may occur, we consider then that few different versions of the software may coexist among all devices. However, we assume that each version of the software is executed by a large number of devices, so that each specific behavior is well represented.

It is assumed that classic security measures have been deployed by the service provider. For instance, network communications between the central server and the devices are already secured and the central server is trusted and managed by the service provider. It is also assumed that traditional security detection mechanisms are deployed on the server side. For instance, the service provider is able to detect that the service is no longer being delivered by the devices.

B. Threat model

The main objective of our approach is to detect significant changes in the behavior of some connected devices among a large population, that may result from potential attacks. We consider that the attacker is not able to modify the probe deployed on each device to collect the Hardware Performance Counters (HPC) values and send them to the central server. We additionally assume that the objective of the attacker is to take control of the compromised devices in order to carry out some attacks on third party systems or to serve his own interests. We also assume that the attacker does not purposely disable or modify the legitimate software of the device, as this attack can already be detected through our previous assumptions. This means that the considered attack consists in installing a malware on the device, and running it along with the legitimate software. This assumption is also realistic because we consider that the attacker wants to hide the malicious activity on the device. Finally, we assume that the attacker cannot corrupt a majority of devices between two consecutive executions of the detection algorithm on the central server.

We argue that an anomaly detection approach based on the analysis of the processor's HPC of the devices is well adapted to the assumptions about the considered environment and threat model. Our approach is designed to be easily implemented and maintained in an industrial context. As described in next sections, this solution results in low overhead on the devices: it only requires minimal modification of the devices' software and reduced network communications. Moreover, the detection approach is based on the comparative analysis of the HPC collected on the devices and does not require the elaboration of a complex model describing the legitimate behavior of the devices.

III. Related works

This section first provides a description of HPC and then discusses some related research works that are mainly based on the analysis of processor's HPC.

HPC store counts of micro-architectural or architectural events of the processor, using a set of special-purpose registers. These registers must be enabled and set up to capture the corresponding events. The number of events that can be monitored ranges from twenty to more than a few hundreds, depending on the processor. They may slightly differ depending on the processor family, but some categories are common to all processor families. Some examples of events are: L1 cache miss, hit, refill, instruction retired, bus access, memory access, branch misprediction, etc. In this paper, we focus on ARM processors which are commonly used for IoT devices.

HPC are generally used for debugging, optimization or profiling purposes, but also in security. Several research works related to malware detection or attack detection using HPC have been developed, but none of them adequately fits the assumptions and threat model described in Section II-B. For example HPCMalHunter [5] uses HPC and Singular Value Decomposition (SVD) to create a real-time behavioral malware detection. This approach applies SVD to HPC traces and then trains an SVM (Support Vector Machine) classifier using benign and malicious programs. However, this approach entirely relies on the modeling of the legitimate application, which we want to avoid in our solution.

The approach presented in [6] consists in modeling legitimate behavior based on HPC using one-class SVM machine learning algorithm, for anomaly detection. However, the environment in which the authors validated the detection algorithm consists of common computers running desktop applications such as Adobe and Internet Explorer, and is not representative of devices with dedicated lightweight operating system and simple applications.

Some research works [7] [8] [9] [10] questioned the use of HPC for security, showing some instability and weak correlations between software execution and the HPCs. For example, Das *et al.* [7] highlights some cases where HPC measurements were inaccurate or not deterministic, due to program execution environment variations (OS, other processes, probe program) and some inconsistency between different types of processors. Zhou *et al.* [10] explored the use of HPC and machine learning to detect malware, showing overall low detection efficiency based on experiments on a large number of programs and malware. However, these experiments were carried out on complex architectures.

In our study, all devices implement a simple and identical architecture, hosting the same OS, processes and probe programs. We have observed in preliminary experiments that the behavior of the type of application investigated in our study is very stable. As we aim to detect remote attacks or misuse of constrained devices, we believe that the HPC should exhibit a significant deviation from legitimate behavior when such attacks are performed. We therefore argue that for such simple devices, the use of HPC to efficiently monitor the behavior of the devices and detect potential anomalies should be more relevant. Our objective is to validate this assumption experimentally.

Some research works have focused on more constrained devices. For example, Wang *et al.* in [11] present a strategy to detect firmware corruption that consists in randomly inserting checkpoints in the firmware code and creating a legitimate signature according to these checkpoints. Such an approach is not relevant under our assumptions as we want our solution to be independent from the software running on the device. Moreover, their approach also needs to create one model per device, and would not be scalable for the monitoring of a large number of devices.

Another relevant approach was proposed by [12] to detect anomalies in real-time multi-threaded cyber-physical systems and Programmable Logic Controllers (PLC). A one-class SVM is learned from the traces from legitimate devices. The anomaly detection algorithm is then run remotely. Two major differences can be highlighted with our work. First, we aim to detect attacks among a large population of devices and second, we do not try to build any prior model of the application processes to minimize any pre-deployment analysis performed on the devices.

To summarize, compared to prior work, our approach targets massively deployed embedded devices sharing the same architecture, with lightweight application software and OS. Unlike existing solutions, our detection approach does not rely on the behavior modeling of the software running on the devices and is based on the comparative analysis of the HPC values collected from the devices using outlier detection algorithms. A thorough experimental analysis is also performed on a large population of devices to validate the efficiency of our approach.

IV. Anomaly detection approach

A. Overview

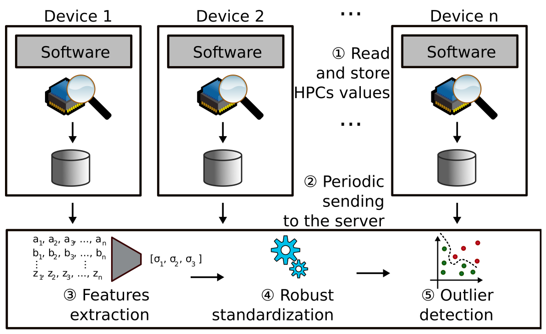
Device 1
Device 2
...
Device n
Software
Software
Software
① Read and store HPCs values
...
② Periodic sending to the server
$a_1, a_2, a_3, \ldots, a_n$
$b_1, b_2, b_3, \ldots, b_n$
$z_1, z_2, z_3, \ldots, z_n$
$[\sigma_1, \sigma_2, \sigma_3]$
③ Features extraction
④ Robust standardization
⑤ Outlier detection

Fig. 1: An overview of the approach

As illustrated in Figure 1, our approach consists in detecting outliers among a fleet of deployed devices, based on the comparative and periodic analysis of the HPC of each device. It is structured into five steps.

On the client side, a probe is embedded in each device to regularly capture and store HPC values in the local file system (step 1). One time series per counter per device is generated, which are then uploaded periodically by each device to the server, through a secure communication channel (step 2), for further processing.

On the server-side, the raw data received from the clients are first processed to extract relevant features (step 3). The feature extraction consists in calculating some descriptive statistics from the data. The next step consists in standardizing the features (step 4) obtained previously to avoid some scaling issues during the execution of the outlier detection algorithm (step 5).

B. Detailed description

In the following, D denotes the set of devices and C the set of observed counters. For a time series or a vector A, $A[i]$ denotes its i-th element.

To collect the HPC, a *probe* software is embedded into each device and captures the values of each counter every Δ_p seconds. These values are stored in a local file (called *trace*) and sent to the server every Δ_s seconds. This parameter allows us to accommodate the possible asynchrony between the devices. This time interval should be calibrated so that the global (or average) behavior during this period is comparable from a device to another.

For a device d, the associated trace is noted T_d and $T_{c,d}$ is the corresponding time series of counter c.Algorithm 1 summarizes the anomaly detection approach.

Algorithm 1: Server-side detection

```
1  for $d \in D$ do
2      for $c \in C$ do
3          for $i$ from $1$ to $|T_{c,d}| - 1$ do
4              $T'_{c,d}[i] \leftarrow (T_{c,d}[i+1] - T_{c,d}[i])/\Delta_p$
5          $U_{c,d} \leftarrow (\max(T'_{c,d}), \mathrm{mean}(T'_{c,d}),$
             $\mathrm{std}(T'_{c,d}), \mathrm{median}(T'_{c,d}))$
6  for $c \in C$ do
7      for $i$ from $1$ to $4$ do
8          $m \leftarrow \mathrm{median}(\{F_{c,e}[i] \mid e \in D\})$
9          $d \leftarrow \mathrm{P}_{95}(\{F_{c,e}[i] \mid e \in D\}) - \mathrm{P}_5(\{F_{c,e}[i] \mid e \in D\})$
10         for $d \in D$ do  $V_{c,d}[i] \leftarrow (F_{c,d}[i] - m)/d$
11 return apply outlier detection algorithm on $V$
```

Instead of using directly the values of the counters represented by the time series $T_{c,d}$, we consider the temporal derivative of this time series, which better reflects the evolution of the activity of the processor. Indeed, the *growth rate* of the events associated to the counters is related to the workload of the processor. This derivative is noted $T'_{c,d}$.

From the raw values of $T'_{c,d}$, we compute the classical statistical features: maximum value (max), mean value (mean), standard deviation (std) and the median value (median). The features of a time series $T'_{c,d}$ is noted $F_{c,d}$ and is defined as:

$$F_{c,d} = (\max(T'_{c,d}), \mathrm{mean}(T'_{c,d}), \mathrm{std}(T'_{c,d}), \mathrm{median}(T'_{c,d}))$$

Depending on the values of Δ_p and Δ_s, $T'_{c,d}$ could be composed of hundreds of values. As we envision a scenario with thousands of devices deployed, extracting statistical features also allows to considerably reduce the size of the data processed by the detection algorithm.

We aim to develop a solution that can be used without recalibration on various software with different behaviors. Thus, we need to "normalize" the data so that the algorithms are effective at various scales. To this end, we standardize the data, i.e. we center and scale them. However, as the data is likely to have a skewed distribution and exhibit outlier values (because of attacks) corresponding to a significant deviation compared to legitimate data, we use a robust version of standardization [13] that replaces using the median and the difference of two percentiles (here the 95th and the 5th, noted P_5 and P_{95}), leading to the following equation:

$$V_{c,d}[i] = \frac{F_{c,d}[i] - \mathrm{median}(\{F_{c,e}[i] \mid e \in D\})}{\mathrm{P}_{95}(\{F_{c,e}[i] \mid e \in D\}) - \mathrm{P}_5(\{F_{c,e}[i] \mid e \in D\})}$$

Each vector $V_{c,d}$ characterizes the behavior of a device. To detect abnormal behaviors, we then use an outlier detection algorithm. It takes as input the set V and returns a subset of D as outliers. The list of anomaly detection algorithms we consider is the focus of the next subsection.

C. Anomaly detection algorithms

There are several algorithms to detect anomalies. We focus on detection algorithms that consist in identifying outliers among a population without any model. We have selected eight unsupervised algorithms for the implementation and experimental validation of our approach: 4 neighbors-based algorithms, 2 clustering algorithms and 2 classification-based algorithms. They are commonly used and are representative of different families of unsupervised machine learning algorithms (see e.g., [14]):

- k-NN (k Nearest Neighbors) [15] is a classifier that can be adapted to outlier detection. In this setting, it classifies an instance as an outlier if it is significantly far from its k nearest neighbors. It has two variants: MeanDIST and KDIST based on the mean distance and the k nearest neighbor, respectively.
- LOF (Local Outlier Factor) [16] classifies an instance as outlier if the local density of this instance is significantly lower than the local density of its k nearest neighbors.
- ODIN (Outlier Detection using Indegree Number) [17] classifies an instance as outlier if it has a low indegree in the k-neighborhood graph.
- k-means is clustering algorithm that searches k centroids. It can be adapted to outlier detection by classifying an instance as outlier if it is far from the centroids. Due to our assumption of homogeneous legitimate devices, we set $k = 1$.

- HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise) [18] [19] is a clustering algorithm. It classifies as outliers all the instances that do not belong to any cluster.
- Isolation Forest [20] is a classification-based algorithm based on random decision trees. An instance is classified as an outlier if it is easily separable, i.e. if the depth of their leaves are small.
- OCSVM (One-Class Support Vector Machine) [21] is a classification-based algorithm based on a SVM model modified to learn from unlabeled instances. We use it in an unsupervised manner: it searches the outliers in a population without any model.

V. Experiments

This section presents the experiments carried out to assess our approach. We first describe the experimental setup and data collection process in Subsection V-A. The different software profiles and the attack payloads used in our experiments are presented in Subsections V-B and V-C respectively. Subsection V-D details the experimental protocol. The following subsections (V-E, V-F and V-G) present the experiments and associated parameters, and discuss the main results.

A. *Set up and data collection*

The experimental platform is made of 100 Raspberry Pi 3B devices with ARM Cortex A53 (ARMv8) processors running Raspberry Pi OS, with a single core activated. Unnecessary services and background processes were removed from the OS. These devices have been chosen and configured to exhibits similar characteristics of real devices deployed in industry.

The Cortex A53 processor allows the monitoring of 28 events. However, there are only 7 HPCs' registers. Among these 7 registers, one is dedicated to record the number of processor cycles (CPU_CYCLES). The other 6 are configurable to monitor any of the 28 events.

Our solution is designed to be application-independent. For this reason, we selected the counters that better reflect the main characteristics of the processor (memory cache, instructions, exceptions, prediction branches, bus access) whose measurement could reveal the presence of a malicious software.

The set of six counters chosen measure the number of refill and the number of accesses on the L1 cache, the number of instructions architecturally executed on the processor, the number of exceptions taken, the number of mispredicted or not predicted branch speculatively executed, and the number of bus accesses performed.

A kernel module, installed in the devices, saves every Δ_p the selected HPC in a local file. Thus, the local file contains time series for each HPC. Every Δ_s, another background process sends these files to the server using FTP and clears the content of the local file. This makes the HPC monitoring independent of the functionality deployed in the devices. In this experiment, we set Δ_s to 1,800s and Δ_p to 5s.

We use the implementations of the *scikit-learn* library [22] for LOF, k-NN, OCSVM, Isolation Forest and HDBSCAN and our own implementation of ODIN and k-means. Source code is available at `https://github.com/PFGimenez/Outlier-detection-from-HPC`.

B. *Software application profiles*

We developed four software applications with different profiles, to validate our approach under different conditions and to evaluate the independence of our detector from the deployed application. The workloads associated to these profiles are designed to be similar to real applications of industrial environment. They use the hardware resources in the same way as an energy management software: they do not overload the platform, perform computations, input/output operations and network communications.

Application profile *S1* corresponds to a device whose main purpose is to collect and store information (such as system logs), and to transmit it periodically. Such a profile corresponds to a regular use of the CPU and network communication and to an intensive use of input/output operations. The second profile *S2* emulates a device that uses few CPU, communications and input/output resources, it could for instance correspond to a simple smart sensor. The third profile *S3* corresponds to a device that executes a more computationally intensive application, such as a device that continuously monitors and reports the energy consumption of a smart building. The last profile *S4* corresponds to a more intensive use of network communications and input/output operations, and may correspond to a gateway for example.

C. *Attack profiles*

To assess the relevance of our approach, we need to evaluate its detection efficiency in the presence of attacks that are representative of actual malicious activities that could be performed on such devices. In our experiments, we have defined different attack profiles that correspond to different attackers' objectives.

Our attacks are either representative of the behavior of some known attacks, such as Mirai [2], or performed by means of real attack tools, such as Hydra[1] or are replays of actual attacks observed on honeypots [23].

Accordingly, we considered eight malicious payloads:

- P1: simulates a DDoS attack from the device;
- P2: a more intensive version of P1;
- P3: simulates a light cryptocurrency mining;
- P4: executes commands drawn from honeypots analysis;
- P5: a more intensive version of P4;
- P6: real attack tool executing a bruteforce SSH;
- P7: the victim of the bruteforce SSH attack;
- P8: payload stressing the CPU.

D. *Experimental protocol*

Our goal is to evaluate the algorithms with a massive amount of deployed devices. To this end, our experiments use populations of 10,000 devices. As we do not own so

[1]Available at https://github.com/vanhauser-thc/thc-hydra

many devices, we sampled traces from 100 devices. Since the devices behave independently, our sampled populations behave like a real population of 10,000 devices.

The sampling is based on two parameters: the population fraction μ that is compromised and the population size n, leading to $(1-\mu)|D|$ legitimate traces and $\mu|D|$ attack traces. Each attack type is drawn equiprobably.

To evaluate the effectiveness of each algorithm described in Section IV-C, we rely on two metrics: False Positive Rate (FPR, the fraction of legitimate devices detected as compromised) and the True Positive Rate (TPR, the proportion of attacks detected).

We then address the following research questions:

- Which algorithms are the most effective, independently of their parameters?
- How the fraction of compromised devices impacts the detection efficiency?
- How well an algorithm calibrated with one software profile performs with data from another profile?
- What is the scalability of these algorithms?

E. Experiments without parameter learning

For each software profile, the experimental protocol is applied considering heterogeneous attacks with 1% compromised devices. Besides the threshold of each outlier detection algorithm, the parameter to set up is the neighborhood size (k) of neighborhood-based algorithms. It is empirically set to 100 for LOF, k-NN KDIST and MeanDIST and 1,000 for ODIN.

We compare the detectors by comparing their ROC (Receiver Operating Characteristic) curves that plot the True Positive Rate against the False Positive Rate). More specifically, we compare their AUC (Area Under ROC Curve), summarized in Table I. All the detectors have high AUC but we can outline some performance differences. LOF, k-NN KDIST and k-NN MeanDIST provide the best results for the four software profiles while HDBSCAN exhibits the worst performance. k-NN MeanDIST and k-NN KDIST generally exhibit similar results and are very similar, so we decided to keep one of them. For these reasons, HDBSCAN and k-NN KDIST are not included in the following results.

TABLE I: AUC of the different algorithms with 1% compromised devices.

ODIN	LOF	MeanDIST	KDIST
0.979	0.991	0.990	0.989

Isolation Forest	k-means	OCSVM	HDBSCAN
0.973	0.981	0.984	0.945

In the next subsection, we analyze the sensitivity of the performance of these algorithms to different parameters.

F. Sensitivity analysis

1) Outlier detection algorithm calibration: Each algorithm assigns a score to each instance: generally, the higher the score, the more likely the instance to be an outlier. Accordingly, one needs to choose a threshold to make a decision.

While this threshold can be constant, one can also use an heuristic to estimate it from the population. The only heuristic we are aware of is for k-NN, described in [17]. This heuristic we call HKF yields poor results in our context because it uses the $\max$ function which is very sensitive to outliers. We slightly modified the heuristic (named HKF+) to use the 95th percentile instead of $\max$. We also propose another heuristic, called α-med. Given a set of distances d (one for each instance) and a parameter $\alpha > 1$, the threshold is defined as $\tau_\alpha(d) = \alpha \operatorname{median}(d)$.

The calibration aims to find optimal values for the parameter of each algorithm. It is carried out with the data of one software profile (legitimate and compromised traces). The parameter is chosen so that FPR is lower than 0.05% and the TPR is as high as possible.

2) Impact of compromised device fraction: We carried out a second experiment with heterogeneous attacks, varying the fraction of compromised devices: 0%, 0.1%, 0.5%, 1%, 2% and 5%. The algorithm parameters have been tuned with 1% compromised devices. Table II reports the average detection efficiency results for each software, when calibrated with one of the three other software. Using another software to calibrate allows us to analyze the robustness of the detection for the calibration process.

First, we can notice that the FPR is generally below 1% and the TPR is around 80%, even though both are highly dependent of the algorithm and the compromised devices fraction. These results show that, with our assumptions and without modelling, outlier detection based on HPC is effective: an algorithm tuned with one software can be successfully applied to another one.

The TPR of all the algorithms is lower with 5% compromised devices. This can be explained by the underlying outliers assumption: they are few and far from each other. However, 5% compromised devices (500 devices here) is a rather large proportion that should not happen instantaneously, so one should expect to detect such attack earlier, when this fraction is lower.

In the following, we only consider the best algorithms: ODIN, LOF, MeanDIST α-med and MeanDIST HKF+.

3) Calibration stability: To study the calibration stability, we reproduce the same experiment, but the parameter learning is performed with the same software profile as the detection. This allows us to compare the FPR and the TPR in the two cases. This experiment was conducted with 1% of compromised devices.

For all four algorithms, the difference in FPR is very small, at most 0.12%. The mean absolute difference of the TPR is below 4% for LOF and MeanDIST α-med and it is 8.00% for ODIN and 16.71% for MeanDIST HKF+. It should be compared to the 80% TPR generally obtained by these algorithms. We can conclude that MeanDIST α-med and LOF are the most suitable algorithms for this problem as their detection capability is very stable on various applications with the same parameter. MeanDIST HKF+ has however a large variability. For this reason, MeanDIST HKF+ is not included in the following experiments.

TABLE II: Detection sensitivity to % compromised devices. Red: TPR ≤50%. Orange: TPR ≤80% or FPR ≥1%.

		0%	*0.1%*		*0.5%*		*1%*		*2%*		*5%*	
Software	Algorithm	FPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR	TPR
S1	ODIN	0.23%	0.18%	72.08%	0.15%	79.82%	0.10%	81.05%	0.06%	45.00%	0.03%	8.54%
	LOF	0.18%	0.09%	78.13%	0.18%	89.43%	0.18%	88.78%	0.19%	86.33%	0.18%	33.37%
	MeanDIST α-med	0.07%	0.07%	77.08%	0.07%	90.32%	0.07%	89.78%	0.07%	89.42%	0.06%	60.81%
	MeanDIST HKF+	0.23%	0.22%	75.42%	0.20%	90.92%	0.15%	88.46%	0.13%	86.56%	0.02%	34.66%
	Isolation Forest	0.79%	0.72%	40.0%	0.34%	53.38%	0.10%	58.18%	0.00%	55.81%	0.00%	28.72%
	k-means	1.61%	0.07%	77.08%	1.58%	64.27%	1.55%	64.94%	1.50%	64.95%	1.50%	65.34%
	OCSVM	0.06%	0.00%	40.63%	0.00%	44.21%	0.00%	36.15%	0.00%	7.80%	0.00%	1.82%
S2	ODIN	0.10%	0.08%	79.38%	0.02%	89.03%	0.01%	77.48%	0.01%	44.01%	0.01%	8.46%
	LOF	0.07%	0.07%	87.50%	0.07%	96.07%	0.07%	94.28%	0.07%	93.05%	0.08%	49.45%
	MeanDIST α-med	0.06%	0.06%	87.08%	0.05%	98.64%	0.05%	98.98%	0.05%	98.59%	0.06%	78.28%
	MeanDIST HKF+	0.10%	0.10%	87.71%	0.10%	97.78%	0.09%	96.79%	0.08%	88.27%	0.00%	55.29%
	Isolation Forest	1.16%	1.01%	39.38%	0.73%	52.72%	0.18%	56.09%	0.01%	53.02%	0.00%	37.33%
	k-means	1.63%	1.64%	42.92%	1.56%	76.39%	1.52%	76.06%	1.46%	75.70%	1.42%	75.79%
	OCSVM	0.03%	0.36%	42.92%	0.00%	41.15%	0.00%	7.35%	0.00%	6.32%	0.00%	1.72%
S3	ODIN	0.03%	0.09%	81.67%	0.02%	93.69%	0.01%	83.64%	0.01%	45.81%	0.00%	14.37%
	LOF	0.04%	0.03%	84.17%	0.02%	94.24%	0.02%	90.64%	0.01%	70.22%	0.01%	42.33%
	MeanDIST α-med	1.02%	0.89%	72.71%	0.50%	100%	0.43%	98.90%	0.28%	98.71%	0.27%	79.81%
	MeanDIST HKF+	0.00%	0.01%	74.79%	0.00%	83.38%	0.00%	77.49%	0.00%	75.49%	0.00%	45.21%
	Isolation Forest	0.42%	0.31%	23.30%	0.09%	73.23%	0.02%	74.98%	0.00%	75.00%	0.00%	56.20%
	k-means	1.60%	1.51%	70.24%	1.22%	78.56%	1.08%	78.04%	0.94%	76.62%	0.91%	75.58%
	OCSVM	0.00%	*no alert*		*no alert*		*no alert*		*no alert*		*no alert*	
S4	ODIN	0.07%	0.07%	80.83%	0.06%	87.88%	0.07%	86.86%	0.06%	49.92%	0.04%	6.30%
	LOF	0.10%	0.10%	80.63%	0.10%	89.18%	0.10%	90.62%	0.10%	89.91%	0.09%	49.44%
	MeanDIST α-med	0.03%	0.03%	68.33%	0.03%	77.60%	0.03%	78.19%	0.03%	76.32%	0.03%	45.97%
	MeanDIST HKF+	0.09%	0.08%	72.50%	0.06%	79.30%	0.05%	75.77%	0.04%	73.91%	0.01%	33.31%
	Isolation Forest	0.08%	0.04%	23.96%	0.02%	49.16%	0.00%	51.33%	0.00%	52.73%	0.00%	43.05%
	k-means	0.00%	0.00%	47.08%	0.00%	53.81%	0.00%	53.60%	0.00%	53.76%	0.00%	53.54%
	OCSVM	0.03%	0.63%	49.17%	0.00%	46.77%	0.00%	35.38%	0.00%	6.25%	0.00%	1.16%

TABLE III: Detection efficiency with 1% compromised devices. Red: TPR ≤50%. Orange: TPR ≤80%.

Software	Algorithm	FPR	TPR P1	TPR P2	TPR P3	TPR P4	TPR P5	TPR P6	TPR P7	TPR P8
S1	ODIN	0.10%	100%	100%	100%	14.50%	100%	100%	33.93%	100%
	LOF	0.18%	100%	100%	100%	79.14%	100%	100%	31.06%	100%
	MeanDIST α-med	0.07%	100%	100%	100%	87.28%	100%	100%	30.92%	100%
S2	ODIN	0.01%	99.80%	99.74%	100%	10.52%	100%	99.09%	100%	99.70%
	LOF	0.07%	100%	100%	100%	54.21%	100%	100%	100%	100%
	MeanDIST α-med	0.05%	100%	100%	100%	91.83%	100%	100%	100%	100%
S3	ODIN	0.01%	99.29%	66.67%	100%	63.09%	66.67%	99.27%	92.63%	81.53%
	LOF	0.02%	99.29%	100%	100%	25.85%	100%	100%	100%	100%
	MeanDIST α-med	0.43%	99.33%	100%	100%	91.83%	100%	100%	100%	100%
S4	ODIN	0.07%	98.61%	100%	100%	53.61%	93.75%	100%	48.88%	100%
	LOF	0.10%	100%	100%	100%	96.18%	100%	100%	28.78%	100%
	MeanDIST α-med	0.03%	100%	100%	100%	36.58%	63.09%	100%	25.87%	100%

4) Impact of attack type: Table III shows the result for 1% compromised devices and specifically the TPR of each attack for the most effective algorithms: ODIN, LOF and MeanDIST α-med. Overall, the attacks are very well detected: P1, P2, P3, P5, P6 and P8 are always successfully detected by MeanDIST α-med. Only two attacks are harder to detect: P4 (command lines drawn from honeypots) on all software profiles and P7 (victim of bruteforce SSH attack) on S1 and S4. The difference concerning P4 and P7 can be explained by the software profile: since S4 involves intensive legitimate network communications, it is harder to notice the commands or SSH bruteforce as they are merged with the legitimate communications. Note that P5, a more intense version of P4, is adequately detected on all software profiles.

G. *Performance overhead and scalability*

Each file sent to the server contains Δ_s/Δ_p values per counter for a total of $|C|(\Delta_s/\Delta_p)$ values. Assuming each counter uses 64 bits, a binary encoding would lead to a file size of $8|C|(\Delta_s/\Delta_p)$ bytes. For a population composed of $|D|$ devices, the server receives and processes about $8|C| \cdot |D|(\Delta_s/\Delta_p)$ bytes every Δ_s. In the context of our experiment (32 bits counters), $|C| = 7$,$|D| = 10000$, $\Delta_p = 5$ and $\Delta_s = 1800$, so about 200MB are processed (i.e. 20kB are sent by each device) every 30 minutes.

All the anomaly detection algorithms we experimented with could process the population of 10,000 devices in less than 15s on a personal computer with an Intel Core i7-3687U CPU @ 3.3GHz processor and 8 gigabytes of RAM. We can estimate the scalability of these algorithms based on their temporal complexity, reported in Table IV. One would expect most of these algorithms (except OCSVM) to be able to process a population of 100,000 devices in less than 5 minutes. Isolation Forest and k-means provided the best execution time performance (about 1s to process the population) and could probably handle 10,000,000 devices in less than 30 minutes. These results show the practical applicability of our approach in a realistic industrial set up.

TABLE IV: Temporal complexity of detection algorithms. $n = |D|$ (number of devices), k is the neighborhood size.

ODIN	LOF	MeanDIST	KDIST
$\mathcal{O}(kn\log(n))$	$\mathcal{O}(kn\log(n))$	$\mathcal{O}(kn\log(n))$	$\mathcal{O}(kn\log(n))$

Isolation Forest	k-means	OCSVM	HDBSCAN
$\mathcal{O}(n)$	$\mathcal{O}(kn)$	$\mathcal{O}(n^3)$	$\mathcal{O}(n\log(n))$

VI. Conclusion and perspectives

In this paper, we explored the relevance of Hardware Performance Counters to support anomaly detection in industrial IoT devices. Then we proposed a lightweight approach based on outlier detection algorithms, well adapted to operational contexts with a massive deployment of devices sharing the same architecture. We have conducted extensive experiments with a large number of devices, various software profiles and attack payloads as well as several outlier detection algorithms. We are not aware of similar experiments in the literature. Our experiments show high detection efficiency as well as high scalability, and our extensive results show that HPC can be successfully used with simple devices in industrial IoT environments. In addition, the calibration of the outlier detection algorithms is not highly dependent on the software program running on the devices, which should facilitate the practical application of the approach when several software updates are performed. For all these reasons, our approach should have an high potential to be successfully deployed in an industrial operational set up. These conclusions need to be validated by other experiments, considering other software workloads and attack payloads, as well as in an industrial environment. To further facilitate the reproducibility of our results, we will make our code and data publicly available.

For future work, more features can be extracted from the counters time series. However we need to keep in mind that we have to compute them regularly for thousands of devices. The detection of network attacks also needs to be improved, either by using complementary existing low level information or by defining new mechanisms more adapted to the monitoring of network communications.

Another perspective consists in taking into account in our anomaly detection approach the evolution of the behavior of the devices over time, instead of comparing their behavior only during the time period when a new measurement is performed.

References

[1] A. Mosenia and N. K. Jha, "A comprehensive study of security of internet-of-things," *IEEE Transactions on Emerging Topics in Computing*, vol. 5, no. 4, pp. 586–602, 2016.

[2] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "DDoS in the IoT: Mirai and other botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[3] K. Khan, A. Mehmood, S. Khan, M. A. Khan, Z. Iqbal, and W. K. Mashwani, "A survey on intrusion detection and prevention in wireless ad-hoc networks," *Journal of Systems Architecture*, vol. 105, p. 101701, 2020.

[4] V. Hassija, V. Chamola, V. Saxena, D. Jain, P. Goyal, and B. Sikdar, "A survey on iot security: application areas, security threats, and solution architectures," *IEEE Access*, vol. 7, pp. 82 721–82 743, 2019.

[5] M. B. Bahador, M. Abadi, and A. Tajoddin, "HPCMalHunter: Behavioral malware detection using hardware performance counters and singular value decomposition," in *2014 4th International Conference on Computer and Knowledge Engineering (ICCKE)*. IEEE, 2014, pp. 703–708.

[6] A. Tang, S. Sethumadhavan, and S. J. Stolfo, "Unsupervised anomaly-based malware detection using hardware features," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2014, pp. 109–129.

[7] S. Das, J. Werner, M. Antonakakis, M. Polychronakis, and F. Monrose, "Sok: The challenges, pitfalls, and perils of using hardware performance counters for security," in *Proceedings of 40th IEEE Symposium on Security and Privacy*, 2019.

[8] V. M. Weaver and S. A. McKee, "Can hardware performance counters be trusted?" in *2008 IEEE International Symposium on Workload Characterization*. IEEE, 2008, pp. 141–150.

[9] V. M. Weaver, D. Terpstra, and S. Moore, "Non-determinism and overcount on modern hardware performance counter implementations," in *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2013, pp. 215–224.

[10] B. Zhou, A. Gupta, R. Jahanshahi, M. Egele, and A. Joshi, "Hardware performance counters can detect malware: Myth or fact?" in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*. ACM, 2018, pp. 457–468.

[11] X. Wang, C. Konstantinou, M. Maniatakos, R. Karri, S. Lee, P. Robison, P. Stergiou, and S. Kim, "Malicious firmware detection with hardware performance counters," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 2, no. 3, pp. 160–173, 2016.

[12] P. Krishnamurthy, R. Karri, and F. Khorrami, "Anomaly detection in real-time multi-threaded processes using hardware performance counters," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 666–680, 2019.

[13] P. J. Rousseeuw and C. Croux, "Alternatives to the median absolute deviation," *Journal of the American Statistical association*, vol. 88, no. 424, pp. 1273–1283, 1993.

[14] V. Chandola and V. K. A. Banerjee, "Anomaly detection: a survey," *Computing Surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

[15] S. Ramaswamy, R. Rastogi, and K. Shim, "Efficient algorithms for mining outliers from large data sets," in *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, 2000, pp. 427–438.

[16] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: Identifying density-based local outliers," *SIGMOD Rec.*, vol. 29, no. 2, p. 93–104, May 2000.

[17] V. Hautamaki, I. Karkkainen, and P. Franti, "Outlier detection using k-nearest neighbour graph," in *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, vol. 3. IEEE, 2004, pp. 430–433.

[18] L. McInnes, J. Healy, and S. Astels, "hdbscan: Hierarchical density based clustering," *Journal of Open Source Software*, vol. 2, no. 11, p. 205, 2017.

[19] L. McInnes and J. Healy, "Accelerated hierarchical density based clustering," in *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE, 2017, pp. 33–42.

[20] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*. IEEE, 2008, pp. 413–422.

[21] M. Amer, M. Goldstein, and S. Abdennadher, "Enhancing one-class support vector machines for unsupervised anomaly detection," in *Proceedings of the ACM SIGKDD workshop on outlier detection and description*, 2013, pp. 8–15.

[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[23] V. Nicomette, M. Kaaniche, E. Alata, and M. Herrb, "Set-up and deployment of a high-interaction honeypot: experiment and lessons learned," *Journal in computer virology*, vol. 7, no. 2, p. 143, 2011.