Hardware Performance Counters for Anomaly Detection in Embedded Devices

Victor Breux and Pierre-Henri Thevenon

Univ. Grenoble Alpes, CEA, Leti, F-38000 Grenoble, France
victor.breux@cea.fr, pierre-henri.thevenon@cea.fr

Abstract. Cyber threats are increasing on a continuous basis, and the Internet of Things (IoT) and embedded devices are also increasingly exposed to attackers. Given the critical nature of these devices in Industrial Control Systems (ICS), the implementation of an Intrusion Detection System (IDS) is necessary. In particular, current researches focus on Host-Intrusion Detection Systems (H-IDS) which leverage signals from the device to detect attacks and intrusions before their impacts become devastating. This paper presents an H-IDS solution that relies on Hardware Performance Counters (HPC) and Machine Learning (ML) anomaly detection to raise alarms when an attack is detected on an embedded device. The proposed solution collects HPCs at the kernel level of the device and uses a remote computer to extract statistical features from the HPC values, then makes a decision with a ML model. An experimental platform simulating an Industrial Control System is presented with a simulated hydroelectric physical process and a hardware emulation of the Programmable Logic Controller managing sensors and actuators logic. The platform also contains 6 typical attacks that often target embedded devices, such as a ransomware, flooding on a communication protocol, SSH bruteforce, malicious data transfers and intensive CPU resources. This setup enables the evaluation of our H-IDS solution under both normal and malicious conditions. Results show that our solution has low CPU and memory footprint on the embedded device and is able to detect critical attacks without raising false alarm.

Keywords: HPCs, machine learning, intrusion detection system, embedded systems, cybersecurity

1 Introduction

As the number of cyberattacks continues to rise, especially in embedded systems, the exclusive use of traditional preventative defences such as root of trust, memory protection or access protection is rarely sufficient. Modern security strategies increasingly integrate Intrusion Detection Systems (IDS) as a complementary tool to detect, in real time, any abnormal behavior that may indicate a compromise in the system. In particular, Host-based Intrusion Detection Systems (H-IDS) have shown considerable promise due to their ability to monitor internal activities during runtime in order to provide fine-grained analysis of the

system behavior.

Unlike network-based IDS (N-IDS), which monitor network traffic to detect a compromise, H-IDS are deployed directly on the host to detect suspicious or unauthorized behavior. H-IDS can be used as a complement of N-IDS to focus on a specific device and to allow the detection of cyberattacks targeting the internal behavior of the device without altering its network interactions. Two approaches are mainly proposed for H-IDS [28]. **Signature-based Detection** method involves identifying known patterns of malicious activity by comparing system information such as byte codes, text strings or instruction sequences with signatures of known attacks stored in their databases. While effective against known threats, this kind of solution is limited in detecting new, modified or obfuscated attack vectors. **Anomaly-based Detection** approach establishes a baseline of normal system behavior and identifies deviations from this reference as potential cyberattacks. By dynamically analyzing the internal system state using internal signals such as log files, system calls, resource usage or even hardware-level measurements, an anomaly-based H-IDS provides unique visibility into the execution of malicious softwares on the target and can detect most attacks such as privilege escalation or code injection. It is particularly suitable for detecting zero-day attacks based on unknown vulnerabilities. A significant challenge in anomaly-based detection is minimizing false alerts while accurately identifying a majority of true threats.
For H-IDS, one emerging reliable data source is Hardware Performance Counters (HPCs). Available in most modern processors (such as Intel, AMD, ARM, RISC-V), HPC registers can be used to measure the number of occurrences of hardware units usage within the processor core. They provide information about the low-level behavior of the processor and count events during process execution related to instructions (e.g. instruction retired, CPU cycles), memory accesses (e.g. cache hits or misses, main memory hits or misses) and the execution behavior on the CPU pipeline, among others. Although modern processors may support monitoring dozens or even hundreds of such events (e.g., 84 on ARM Cortex-A72, 62 on Cortex-A53), a key hardware limitation is that only a limited number (typically between 2 and 8) can be monitored simultaneously as only a limited number of CPU registers are allocated for HPC monitoring. This limitation requires a careful selection of HPC.

Originally designed for performance profiling and debugging, HPCs have been leveraged in recent years for security purposes, particularly for anomaly detection. Some studies demonstrated the efficiency of HPCs in detecting microarchitectural attacks [21,26,25,20], control flow hijacking attacks [12,35,38,10,24], rootkits [34,30], malwares [27,29,15,19] and some specific malwares types such as ransomwares [23,37,8,36] or DDoS [22]. H-IDS solution based on HPCs have been criticized because of the non determinism of HPC registers [11], however they are suited for embedded systems, such as Programmable Logic Controllers (PLC) or Industrial IoT (I-IoT) gateways, where the normal behavior is usually deterministic, not too complex and well defined.

Contributions: In this paper, we provide an innovative solution leveraging HPCs for H-IDS on embedded systems. Our contributions are as follows:

1. We present a novel H-IDS for embedded systems, using HPCs to detect malicious behavior in an embedded device. Our approach combines statistical features extraction with a reconstruction-based AI model.
2. We implement a complete and operational prototype on an ARM-based embedded device integrating custom kernel module for HPC extraction and offloading the anomaly detection process to a remote server.
3. We design a realistic hardware-in-the-loop experimental platform comprising embedded systems (PLC and I-IoT gateway) that are representative of an hydroelectric industrial control system. This testbed allows the execution of both benign and malicious behaviors and serves for the dataset generation. We also define and implement a representative set of attacks commonly targeting embedded systems.
4. We thoroughly evaluate the performance overhead introduced by our solution by analysing its impact on the CPU usage, the memory consumption, the network traffic and the inference duration.

Outline: The rest of the paper is organized as follows. Section 2 provides a detailed description of the complete solution and its implementation. Section 3 then focus on describing the methodology used to identify the best HPCs, the extraction of features and the machine learning model for anomaly detection. Section 4 introduces a dedicated tested with different malware behaviors and evaluates the solution. Section 5 reviews the state-of-the-art of H-IDS based on HPCs for embedded systems. Finally, Section 6 concludes with a summary and future directions.

2 Proposed solution

As illustrated in Figure 1, the proposed solution described in this section is a countermeasure designed to monitor and detect abnormal behaviors in an embedded device using HPCs. It is based on a centralized architecture, where the storage and analysis of the raw data extracted from all embedded devices is handled on a remote server ensuring easier management and scalability. The target device is equipped with a kernel module and a dedicated application to extract HPC data in real-time, while the remote server is responsible for analyzing these data, calculating anomaly scores through the machine learning (ML) model, and displaying the results via a comprehensive dashboard.

2.1 Embedded solution

The target device integrates the two main components required for HPC extraction and transmission: a kernel module and a user-space application.

The kernel module is designed to interact directly with the registers of the Hardware Performance Counters to periodically extract low-level statistics in

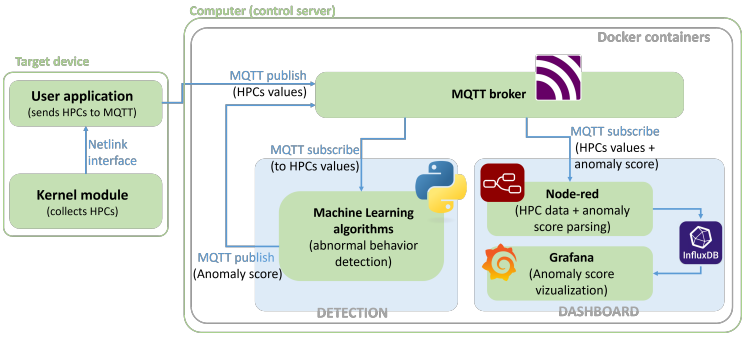

Fig. 1. Architecture of the solution

each core of the processor using assembly code. For each specific counter, its global value is obtained by summing the measurements from all cores. By reading these counter values, the kernel module captures real-time data about the system's resources usage, thus enabling an understanding of its hardware behavior. Compared to a user-space application for HPC extraction, the kernel-level extraction mechanism limits the impact of system load and scheduling for real-time monitoring and allows us to collect HPC raw values at 100Hz (every 10 ms). We chose 100Hz as a good trade-off between a sufficiently high sampling rate and a low overhead induced by the extraction.

The user space application aggregates the raw data, formats them and prepares them for a transmission to the centralized remote server. The application communicates with the kernel module through Netlink, a communication protocol to exchange data between kernel and user space through a low latency channel. The processed data are sent to the centralized served through MQTT (Message Queuing Telemetry Transport), a lightweight protocol allowing communications with limited bandwidth.

2.2 Remote server: data analysis and client application

The function of the remote server is to receive, analyze and visualize data transmitted from the embedded device. The client application is at the core of this component, which analyzes the incoming raw HPC data, preprocesses them to compute statistical features and finally calculates the anomaly scores using the trained ML model. These algorithms are designed to detect subtle deviations in the target system's behavior, which may indicate a malfunction or a potential security threat. A dashboard also allows to visualize both the detection results (anomaly scores) and HPC values.

The server's architecture relies on a network of Docker containers deployed on the client machine to provide a modular and scalable environment and facilitate the setup of the solution. The components deployed in each container are described below:

- to facilitate the communication between the embedded devices(s) and the remote server, the system is based on MQTT protocol. The Mosquitto **MQTT broker** acts as an intermediary to collect published messages from multiple devices and transmits them to their subscribers.
- the **Node-RED server** subscribes to MQTT services and process incoming data packets from embedded devices. It parses these data packets into individual HPC register values in order to write them in an InfluxDB database for further analysis.
- the **client** application performs the computation of statistical features, computes the anomaly score with the ML model and publishes it on the MQTT channel.
- **InfluxDB** is a time-series database and serves as a central repository for HPC raw values and the calculated anomaly scores, providing an easy-to-query backend for data analysis or display.
- A **Grafana dashboard** is used to visualize the anomaly scores computed by the ML model and sorted in the InfluxDB database.

3 Anomaly detection methodology

An overview of the methodology is described in Figure 2. First, relevant HPCs were selected and their signals were preprocessed. Then, on those preprocessed signals, features were computed on sliding time windows and used as inputs for the ML anomaly detection model.

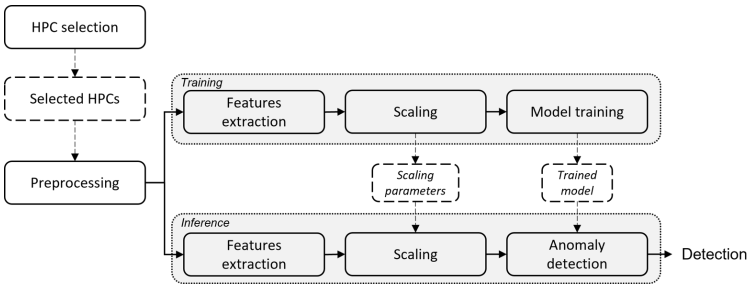

Fig. 2. Overall methodology pipeline

3.1 HPCs selection and preprocessing

From all the HPCs available on a processor, only a limited number (let say m) can be monitored concurrently. A first step of HPC selection was thus required to select these m HPCs. Our selection method relied on mutual information [14]. Mutual information between two random variables is a measure from the information theory that quantifies the dependency between them. A higher mutual information value represents higher dependency.

To select the best m HPCs, several datasets were generated (see Sections 4.1 and 4.2), each comprising m HPCs with HPC sets differing across datasets, allowing to have recordings for all available HPCs. Based on these recordings, the mutual information between each HPC and the ground-truth label (0 for benign and 1 for an attack) was computed with a non-parametric method relying on entropy estimation from k-nearest neighbors [17]. This approach gave us the value of mutual information for all HPCs available on the processor. We finally selected the m non-correlated HPCs with the highest mutual information value and a final dataset with this set of HPCs was generated and used for the rest of the study. The corresponding signals were then preprocessed with two additional steps :

- resampling : the signals were resampled to match the 100 Hz sampling frequency of the HPC extraction. The missing values induced by the resampling were filled with the value following the gap evenly distributed on it. This step is necessary to ensure a constant time gap between consecutive values for correct features computation, and to ensure that no accumulation of HPC values occurs if some time slots were missing.
- smoothing: Exponentially-Weighted Moving Average (EWMA) technique was applied on the signals to smooth them and remove noise. The weighting parameter was arbitrarily set to $\frac{2}{9}$ (corresponding to a span of 8 for the filtering).

These preprocessed signals were used for the next steps of features computation and anomaly detection model.

3.2 Features computation

After selecting the HPCs and preprocessing the signals, a set of statistical features was computed. For each HPC, a sliding window of 1 second (i.e. 100 measurements) was moved along the signals and features were computed on each window. On each window, and for each HPC, two signals were considered for features computation :

- raw signal : HPC signal after the preprocessing step
- derivative signal : derivative of the HPC signal, computed using the second order central differences

On both the raw signal and its derivative, for each time window and each HPC, the following features were computed : mean, standard deviation, minimum, median, maximum, root mean square, median absolute deviation from the median, kurtosis, skewness and coefficient of variation.

In total, on each time window, $20 \times m$ features were computed (10 on the signal and 10 on its derivative for each of m HPC) to represent the trend and the variation of the signals and their derivative and to capture distribution information over 1 second time windows [13]. Computed features were then scaled with a Min-max scaling to ensure they were all in comparable ranges before being fed to the ML model.

3.3 ML anomaly detection model

For anomaly detection we chose to use a one-class anomaly detection model based on machine learning. The anomaly detection model was trained only on the features from the normal behavior of the system and any deviation from this behavior should be detected as an anomaly.
Our method was based on a reconstruction method with an autoencoder (AE) and is described in Figure 3.

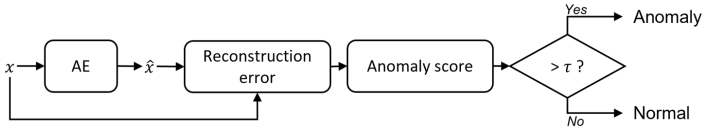

Fig. 3. Reconstruction-based anomaly detection with an autoencoder (AE) and a threshold τ

An autoencoder is a neural network architecture whose aim is to reconstruct the input (mimic the identity function) from a compression (or latent vector) of the input. After training, the reconstruction error of x, $r(x)$, was computed as the L2-distance between x and its reconstruction by the autoencoder.

The autoencoder was trained to reconstruct data from the normal behavior in the training set. Therefore a data not coming from the training distribution would be poorly reconstructed with a high reconstruction error.
A soft-decision anomaly score s was computed with the sigmoid function σ and the relative difference between the reconstruction error $r(x)$ and t, the maximum of the reconstruction errors on the training set :

$$s(x) = \sigma\left(2 * \frac{r(x) - t}{t}\right) \tag{1}$$

The higher the reconstruction error, the higher the anomaly score; and reconstruction errors greater than t gave anomaly scores above 0.5. This anomaly score is between 0 and 1 and can be interpreted as the probability of the vector x to be an anomaly with respect to the training set distribution.

Finally, a predefined threshold $\tau \in [0, 1]$ was used on the anomaly score to have a binary decision (1 being the anomaly label and 0 the normal label) :

$$A(x) = \begin{cases} 1, & \text{if } s(x) \geq \tau. \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

Increasing τ would decrease false alarms as higher deviation from the maximum reconstruction error on the training set would be needed to be detected as an anomaly.

4 Experimental results

4.1 Evaluation platform

Our experimental platform in Figure 4 is mainly based on the STMicroelectronics evaluation board STM32MP157F-DK2 [5] running an embedded Buildroot distribution with a SLTS (Super Long Term Support) Linux kernel 6.1. Its processor is widely used in embedded devices with high resources requirements, such as I-IoT gateways or PLC, and has two ARM Cortex-A7 cores running at 800 MHz. Each core of the processor provides 42 HPCs and enables the extraction of measurements from four registers simultaneously, in addition to CPU cycles register. As discussed in Section 2, we extracted measurements from the HPC registers every 10 ms using a dedicated kernel module to extract the values and to limit the performance overhead.

We implemented software scripts on this platform to reproduce the internal behavior of off-the-shelf industrial embedded systems. In our case, I-IoT and PLC functionalities were considered as the normal behavior of the system. The PLC abstraction executes control logic using OpenPLC [7] to drive a hydroelectric simulated physical process using Modbus TCP protocol; it also communicates with a SCADA server that displays the state of each sensors and actuators. The I-IoT gateway extracts data sensors from a Raspberry Pi Sense HAT add-on board using I^2C communication bus and sends the values to an I-IoT server.

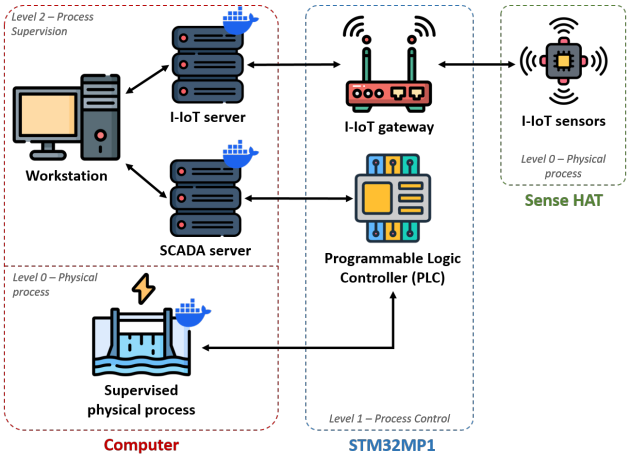

Fig. 4. Evaluation platform

The goal of our detection solution is to detect malicious modifications of the system and/or malicious applications aiming at disrupting the availability, integrity or confidentiality/privacy of target applications and their data. Applicative malwares were developed to stress the target system and to evaluate

the detection mechanisms, and were written from scratch so they could be updatable and expandable.
Even if I-IoT devices or PLC are not connected to the Internet, they are connected to Information Technology (IT) systems or supervision computers within an internal network. The attacker can gain access to the IT systems thanks to vulnerabilities or misconfigured network defenses that then allows connecting to the internal network, and thus to the target device. However in this work, we mainly focused on the main impacts of a malicious behavior on the device, so we considered that the attacker is already connected to the target system with super-user privileges.

*Malware 1 - Data transfer to attacker (**M1**):* This malware aims to mimic the attacker sending a memory dump with potentially confidential data to its attacker's server. It is based on a FTP client that sends all files from a defined folder to the FTP server.

*Malware 2 - Task execution with data leakage (**M2**):* This malware aims at running a heavy resource consuming application on the target system. We chose a cryptocurrency mining use-case as such malwares were infecting many real embedded devices in the past years. The mining is performed by cpuminer software [1] on the target system and a Stratum server.

*Malware 3 - Cryptolocker (**M3**):* This malware aims at mimicking the behavior of a classic cryptolocker, a type of ransomware that encrypts either a file or a directory making its content inaccessible. This malware connects to the target system via SSH and then encrypts the target directory using ccrypt, a Linux utility for file encryption and decryption.

*Malware 4 - Modbus TCP flooding (**M4**):* A significant threat to industrial control systems (ICS) is the denial-of-service (DoS) attack on industrial communication protocols, which can lead to the disruption or corruption of the associated physical processes. One such attack involves Modbus flooding, where the attacker sends a high volume of requests to a Modbus TCP server, overwhelming its capacity.

*Malware 5 - SSH bruteforce (**M5**):* Dictionary based attacks on telnet or SSH services are one of the most common vectors of attacks in I-IoT devices. The implemented attack is based on Hydra [2], a widely used password-cracking tool that can perform brute-force or dictionary-based attacks.

*Malware 6 - Communication DoS (**M6**):* This malware aims at blocking communications between the application OpenPLC running on the target to execute control logic and the rest of its environment (namely the SCADA server and the physical process simulator). We add an iptables rule on the target system that blocks incoming and outgoing communication on TCP port associated with Modbus TCP.

Evasivity: Strategies to soften the impact of the malwares on the device were also developed and are presented in Table 1. Malwares **M4** and **M6** do not integrate such strategies. These strategies allow to lower the visibility of the malwares by detection mechanisms.

Table 1. Evasivity of malwares

Malware	Name	Evasivity
M1	Data transfer to attacker	Files are transferred in decreasing order of size
M2	Task execution with data leakage	Ten executions of cpuminer with evasivity parameter from 0 to 100 for the last execution
M3	Cryptolocker	Files are encrypted in decreasing order of size with a sleeping time in between
M5	SSH bruteforce	Hydra is executed with a decreasing number of threads

4.2 Dataset

Applications: The dataset with the selected HPCs contained executions of several applications :

- Normal behavior (**N**) : nominal and safe behavior of the system with OpenPLC application handling the PLC logic of the physical process. Data from this application were labelled as 0 (benign state) and represented a recording of 1 hour.
- Succession of malware applications : all malwares described in Section 4.1 were sequentially executed on top of the normal behavior. Data obtained during malware executions were labelled as 1 (anomaly state). Some malwares were executed with evasivity strategies as described in Table 1.

Training and evaluation split for ML model: The dataset was divided into 2 parts for the training and the evaluation of the ML anomaly detection model respectively. The training set contained only data from the benign state of the system, so that the model could learn the nominal distribution and detect any deviation from it as an anomaly. This set was used to train the anomaly detection model and was made of the first 80% of the normal behavior recording (corresponding to a contiguous recording of 2880 seconds).
Conversely, the goal of the evaluation set (also called test set) was to evaluate the trained detection model on new data that were not seen during training. This allowed assessing model's performances and its ability to generalize on new data. Evaluation metrics (see Section 4.3) were computed on this set. The remaining 20% of the normal behavior recording was incorporated into the test set (corresponding to a contiguous recording of 720 seconds), in conjunction with all entire malware recordings.

4.3 Anomaly detection results

As explained in Section 3.1, we selected HPCs based on mutual information with the label. The five selected HPCs, that had the highest dependency with the label, were :

- DATA_WRITE_INSTR: Counts the number of data write instructions accepted by the Load Store Unit.
- Data Memory Access: Counts the number of read or write instructions in the data memory.
- Bus access write: Counts the number of bus accesses for write instructions.
- Bus Cycles: Counts the number of clock cycles used for write or read instructions on bus.
- CPU cycles: Counts the number of clock cycles of the CPU.

Selected HPCs signals were resampled and smoothed as discussed in Section 3.1. A sliding window of 100 timesteps was then used to extract 1 second time windows and compute features (see Section 3.2). On the training set, the sliding window moved with a step of 1 to augment the number of windows during training. However, the window moved with a step equal to the window size on the test set to have non-overlapping windows for the evaluation resembling a real-time scenario. On each window, 20 features were computed for each of the 5 HPC, resulting in features vectors with 100 components.

The encoder and the decoder of the AE model both had 3 fully-connected hidden layers (with Exponential Linear Unit activation) with 128, 64 and 32 units for the encoder and 32, 64, 128 for the decoder. The output layer of the encoder had 16 units (with *tanh* activation) and the output of the decoder had 100 units (=size of the features vectors). Dropout regularization was also used during training with a rate of 0.2. The AE model was trained on the scaled features vectors from the training set during 50 epochs on batches of size 512 using Adam optimizer and Mean-Squared Error loss function.

Evaluation metrics We evaluated the binary detection with True Positive Rate (TPR) and False Positive Rate (FPR) metrics computed on the test set. These two metrics represent respectively the proportion of time windows from malware executions correctly labelled as anomalies by the model and the proportion of time windows from the benign application misclassified as anomalies. TPR should be maximized, whereas FPR should be as low as possible. In our use case, decreasing the FPR is crucial because a detection system with too many false alarms would be unusable.
We also evaluated the model by computing the detection accuracy on each application (Acc_a for application a) in the test dataset. Accuracy is the proportion of time windows correctly classified by the model. In our case, accuracy on an application is the proportion of the application execution time correctly predicted.

Table 2. Evaluation of anomaly detection model on the test set

	TPR	FPR	$\text{Acc}_{\mathbf{M1}}$	$\text{Acc}_{\mathbf{M2}}$	$\text{Acc}_{\mathbf{M3}}$	$\text{Acc}_{\mathbf{M4}}$	$\text{Acc}_{\mathbf{M5}}$	$\text{Acc}_{\mathbf{M6}}$
$\tau = 0.4$	42.04 %	0.14 %	2.07 %	73.18 %	22.57 %	100 %	41.46 %	0.83 %
$\tau = 0.5$	41.19 %	0.14 %	1.67 %	70.25 %	18.96 %	100 %	41.33 %	0.83 %
$\tau = 0.6$	40.35 %	0 %	1.67 %	67.32 %	16.49 %	100 %	41.03 %	0.56 %

Results The binary decision computed by the model with Equation (2) was evaluated for 3 values of threshold τ : 0.4, 0.5 and 0.6. Values of the metrics for those 3 thresholds are given in Table 2.

Anomaly scores computed by the model on the test set, using Equation (1), for each application are shown in Figure 5.

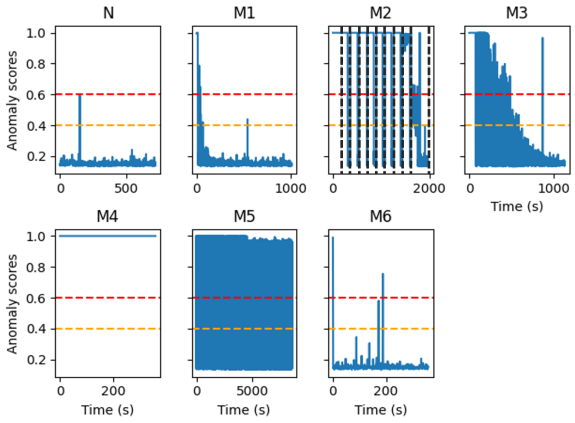

Fig. 5. Anomaly scores on the test set for each application (vertical dashed lines in **M2** separate different executions of the same application). Orange line indicates the warning threshold and red line the alert threshold

The results in Table 2 show that increasing the threshold reduced the FPR from 0.14% with a threshold of 0.4 to 0% with a threshold of 0.6. TPR was slighlty reduced when increasing the threshold : from 42.04% with 0.4 to 40.35% with 0.6.

The low TPR and accuracy values on malware could be explained because a malware execution does not contain only malicious portions. Indeed, a malware could also perform safe operations in addition to its malicious payload. Detecting the entire execution of an attack as abnormal is not necessary, we only need the model to detect at least one portion (even small) of the attack execution to

consider it as detected. Therefore the TPR and the accuracy on malwares can be low without affecting the model capacity of raising alarms during attacks.

For our solution, we chose to use 0.4 as a warning threshold (in orange on Figure 5) that indicated when something needed attention without necessarily being detected as an attack; and 0.6 as the alert threshold (in red on Figure 5).

Based on results from Table 2 and Figure 5, we can give more details on the model's detection performance on each malicious application :

- data transfer (**M1**) : only the beginning of the attack where larger files were sent raised an alarm. The evasivity strategy of the attack allowed it to bypass the detection by not raising alarms when smaller files were being sent.
- CPU miner attack (**M2**) : The model detected anomalies in each of the 10 executions (separated by the black-dashed lines on Figure 5) of this attack (with 67.32% detection). As explained before and since at least one portion in each execution raised an alarm, we consider this attack as well detected by the model.
- Cryptolocker (**M3**) : 16.49% of the attack execution was detected as abnormal by the model and Figure 5 shows that the anomaly scores decreased during the execution. As for the data transfer attack, dealing with small files did not raise alarm and bypassed the model.
- Modbus TCP flooding (**M4**): this attack execution was entirely detected, with 100% accuracy and anomaly scores constantly above the alert threshold.
- SSH bruteforce (**M5**) : almost half of the attack execution was detected (41.03%). The model raised alarm during this attack making it well detected.
- Communication DOS (**M6**) : only a poor proportion of the attack execution was detected, however the model still raised some alarms during the execution.

4.4 Overhead results

In this section, we evaluated the performance overhead introduced by our solution, focusing on memory usage, CPU consumption and network traffic. Understanding these impacts on an embedded device is mandatory for determining the feasibility of deploying such solution in a resource-constrained environment.

Impact on the memory and CPU resources: To measure the overhead on resource consumption, global memory and CPU consumption were continuously monitored on the platform over two-hour periods, both in the presence and absence of the detection solution. The memory and CPU usage remained stable and comparable to that of the reference consumption. This indicates that our solution did not introduce significant additional memory demands, making it suitable for systems with limited memory resources.

Impact on the network: Our embedded solution transmitted raw HPC to the control server at a rate of 2976 bytes per second using MQTT protocol. These 2976 bytes corresponded to the MQTT frame used to transmit one hundred sets of five 32-bit HPC registers and 64-bit timestamp value. This transmission rate is manageable for most network infrastructures; however, in bandwidth-constrained environments, this could contribute to network congestion.

Anomaly detection inference time: We evaluated the time to make an inference on a 1 second HPC time window. The inference time comprised the preprocessing of the signals, the features computation and the detection by the ML model. Evaluated on a Intel Core i5-1235U with 10 cores at 1.3 GHz, the mean inference time for one inference was about 73 ms (with a standard deviation of 23 ms).

In summary, our anomaly detection solution demonstrated a low memory and CPU footprint and a manageable network usage. These findings suggests that the solution is viable for deployment in embedded systems with limited resources. However, for applications with stricter performance and real-time requirements, further optimizations may be necessary. One solution would be to implement the ML model in the embedded device to reduce the network bandwidth usage; however, this approach would result in increased CPU and memory consumption.

5 Related works and discussions

Numerous approaches were proposed in the state-of-the-art to detect and mitigate attacks targeting embedded devices such as PLC or I-IoT gateway. Main characteristics of these approaches are summarized in Table 3, with a comparison to our proposed solution.

As illustrated in Table 3, two different approaches were adopted in the scientific literature. The first approach involved the use of numerous real-world malwares, taken from public malwares library, and benign samples to measure the accuracy of their detection algorithms. The second approach involved forged attacks to mimic potential impacts of an attack in order to detect all deviations from the reference behavior of the system. If the first approach evaluated the solution with a greater library of normal and malicious executables, each sample was evaluated individually in a sandbox, which was not an accurate representation of a real-life scenario. At the opposite, forged attacks added more realism in the malwares impacts as they were specifically developed to target a real-life scenario such as the behavior of a PLC or a data gateway.

Another important consideration in evaluating anomaly detection methods is the granularity of analysis of the samples. Some approaches operated without appropriate timing windows, triggering alerts for each individual extracted sample or for short timing windows. While these solutions offer a better sensitivity and detection speed, they also increased the potential number of false alarms. For example, a seemingly low FPR of 0.1% may become problematic when applied at high detection rate such as 10ms as it would introduce a potential false

Table 3. State-of-the-art of H-IDS based on HPC for embedded devices

Method	Sampling rate [window]	HPC extractor	Target	Attacks	Overhead	Detection
[18]	1kHz [250ms]	PAPI	PLC Wago Raspberry Pi	forged (6,no)	CPU: 10% Mem: 5%	Off
[9]	0.2Hz [1800s]	kernel module	Raspberry Pi 3b	forged (8,no)	N/A	On, R
[19]	1kHz [5-45 ms]	perf	Raspberry Pi 3b	malwares	N/A	Off
[31]	N/A	perf	Raspberry Pi 3b+	forged (1,no)	N/A	Off
[33]	1kHz	perf & simperperf	Google Pixel 4	malwares	execution time: 3.2%	On, L
[16]	1kHz [0.25s]	PAPI	Raspberry Pi	forged (3,no)	N/A	Off
[6]	1kHz	perf	Raspberry Pi 2b	malwares	N/A	Off
[32]	1kHz [100ms]	hardware	Xilinx ARTIX-7	forged (7,no)	hardware implementation	On, R
Our method	**100Hz [1s]**	**kernel module**	**STM32MP1**	**forged (6,yes)**	**0%**	**On, R**

Sampling rate [window]: sampling rate and timing window used by the solution
HPC extractor: method used to extract the HPC values
Target: target device used for the evaluation
Attacks: type of attacks with the number of attacks and the consideration of evasiveness for forged attacks
Overhead: performance overhead on the target device
Detection: Online (On) represents a real-time detection and Offline (Off) a detection at a later time. Local (L) means a detection on the targeted device and Remote (R) on a distant server
N/A: no information provided by the authors

alert every 10 seconds making the solution unusable. Conversely, solutions with equivalent FPR yet with suitable timing windows yield a reduced frequency of false alarms. Identifying the best compromise between detection accuracy, overhead and the number of false alerts is crucial before deploying such detection solution in real use-cases. In our solution, we chose a sampling rate of 100Hz and a timing window of 1 second (i.e. 100 measurements) which seemed the best compromise between these metrics.

Evasive attacks, which are designed to bypass detection systems by lowering their impact on the target, were rarely integrated to evaluate the accuracy of a detection method in the literature. In our evaluation, we proposed different

evasive strategies such as increasing the sleep time between task execution or decreasing the file size for a datalogger or a cryptolocker attacks. Our results highlighted the difficulty to detect correctly an attack if its malicious payload was hidden between the normal process of the system.

Despite significant progress in the field of attack detection using hardware performance counters, very few papers proposed a comprehensive and real-time solution. Most of the solution described in the state-of-the-art were based on tools to extract the raw HPC values such as PAPI Library [3] or perf [4] that seem not appropriate for a real solution. Our solution was based on dedicated kernel module to extract HPC data and all firmware to send these data from the kernel space to a remote server in order to compute the anomaly score. Using a kernel module enhances security and reduces overhead compared to user-space solution but it reduces portability across different CPU architectures.

6 Conclusion

In this work, we proposed a novel H-IDS leveraging Hardware Performance Counters (HPCs) for anomaly detection on embedded devices. Our system extracts HPC values at the kernel level and processes them using statistical features on 1-second time windows. An autoencoder-based model is then used to identify deviations from the learned normal behavior of the system.
To support the evaluation of our approach, we introduced a realistic experimental platform simulating a physical process controlled by OpenPLC. This setup allowed us to emulate industrial control systems and to test the robustness of our H-IDS against a set of common and evasive attacks targeting embedded devices. Our results demonstrate that the proposed method achieves low overhead and no false alarm, while successfully detecting various types of attacks, such as Modbus TCP flooding, SSH brute-force, and CPU miner activity. However, some highly evasive attacks, such as data exfiltration and cryptolockers, were able to evade detection, highlighting current limitations of the solution.
Future work will focus on expanding the attack set and improving the realism of the adversarial behaviors by integrating more advanced evasiveness parameters. Additionally, we plan to embed the ML model directly into the monitored device to remove the remote communication. This would reduce network impact but introduce new challenges related to resource usage and potential self-interference, as the monitoring process itself may affect the HPCs. We also envision enhancing the anomaly detection capabilities by incorporating complementary signals, such as system call traces, to provide a more holistic view of system behavior.

Acknowledgements

This work was supported by the French National Research Agency in the framework of the "Investissements d'avenir" program (IRT Nanoelec, ANR-10-AIRT-05) and under the France 2030 label (Superviz ANR-22-PECY-0008). The views reflected herein do not necessarily reflect the opinion of the French government.

References

1. Cpuminer-multi. https://github.com/tpruvot/cpuminer-multi, accessed: 2025-04-25
2. Hydra. https://www.kali.org/tools/hydra/, accessed: 2025-04-25
3. Papi: The performance application programming interface. https://hpc.llnl.gov/software/development-environment-software/papi-performance-application-programming-interface, accessed: 2025-04-25
4. perf: Linux profiling with performance counters. https://perfwiki.github.io/main/, accessed: 2025-04-25
5. Stm32mp157f-dk2 – discovery kit with stm32mp157f mpu. https://www.st.com/en/evaluation-tools/stm32mp157f-dk2.html, accessed: 2025-01-21
6. Adhikari, S., Asad, H., Jones, K.: Enhancing IoT Security: Novel Mechanisms for Malware Detection using HPCs and Neural Networks. In: 2023 IEEE 22nd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom). pp. 1455–1463. IEEE, Exeter, United Kingdom (Nov 2023). https://doi.org/10.1109/TrustCom60117.2023.00199, https://ieeexplore.ieee.org/document/10538828/
7. Alves, T.R., Buratto, M., de Souza, F.M., Rodrigues, T.V.: Openplc: An open source alternative to automation. In: IEEE Global Humanitarian Technology Conference (GHTC 2014). pp. 585–589 (2014). https://doi.org/10.1109/GHTC.2014.6970342
8. Anand, P.M., Charan, P.V.S., Shukla, S.K.: HiPeR - Early Detection of a Ransomware Attack using Hardware Performance Counters. Digital Threats **4**(3), 43:1–43:24 (Oct 2023). https://doi.org/10.1145/3608484, https://dl.acm.org/doi/10.1145/3608484
9. Bourdon, M., Gimenez, P.F., Alata, E., Kaaniche, M., Migliore, V., Nicomette, V., Laarouchi, Y.: Hardware-Performance-Counters-based anomaly detection in massively deployed smart industrial devices. In: 2020 IEEE 19th International Symposium on Network Computing and Applications (NCA). pp. 1–8. IEEE, Cambridge, MA, USA (Nov 2020). https://doi.org/10.1109/NCA51143.2020.9306726, https://ieeexplore.ieee.org/document/9306726/
10. Das, S., Chen, B., Chandramohan, M., Liu, Y., Zhang, W.: ROPSentry: Runtime defense against ROP attacks using hardware performance counters. Computers & Security **73**, 374–388 (Mar 2018). https://doi.org/10.1016/j.cose.2017.11.011, https://linkinghub.elsevier.com/retrieve/pii/S0167404817302481
11. Das, S., Werner, J., Antonakakis, M., Polychronakis, M., Monrose, F.: Sok: The challenges, pitfalls, and perils of using hardware performance counters for security. In: 2019 IEEE Symposium on Security and Privacy (SP). pp. 20–38 (2019). https://doi.org/10.1109/SP.2019.00021
12. Demme, J., Maycock, M., Schmitz, J., Tang, A., Waksman, A., Sethumadhavan, S., Stolfo, S.: On the feasibility of online malware detection with performance counters. ACM SIGARCH Computer Architecture News **41**(3), 559–570 (Jun 2013). https://doi.org/10.1145/2508148.2485970, https://dl.acm.org/doi/10.1145/2508148.2485970
13. Fulcher, B.D.: Feature-based time-series analysis (Oct 2017). https://doi.org/10.48550/arXiv.1709.08055, http://arxiv.org/abs/1709.08055, arXiv:1709.08055 [cs]
14. Guyon, I., Elisseeff, A.: An introduction to variable and feature selection. J. Mach. Learn. Res. **3**(null), 1157–1182 (Mar 2003)

15. Kadiyala, S.P., Jadhav, P., Lam, S.K., Srikanthan, T.: Hardware Performance Counter-Based Fine-Grained Malware Detection. ACM Transactions on Embedded Computing Systems **19**(5), 1–17 (Sep 2020). https://doi.org/10.1145/3403943, https://dl.acm.org/doi/10.1145/3403943
16. Konstantinou, C., Wang, X., Krishnamurthy, P., Khorrami, F., Maniatakos, M., Karri, R.: HPC-Based Malware Detectors Actually Work: Transition to Practice After a Decade of Research. IEEE Design & Test **39**(4), 23–32 (Aug 2022). https://doi.org/10.1109/MDAT.2022.3143438, https://ieeexplore.ieee.org/document/9681697/
17. Kraskov, A., Stögbauer, H., Grassberger, P.: Estimating mutual information. Physical Review. E, Statistical, Nonlinear, and Soft Matter Physics **69**(6 Pt 2), 066138 (Jun 2004). https://doi.org/10.1103/PhysRevE.69.066138
18. Krishnamurthy, P., Karri, R., Khorrami, F.: Anomaly Detection in Real-Time Multi-Threaded Processes Using Hardware Performance Counters. IEEE Transactions on Information Forensics and Security **15**, 666–680 (2020). https://doi.org/10.1109/TIFS.2019.2923577, https://ieeexplore.ieee.org/document/8737990/
19. Kuruvila, A.P., Karmakar, S., Basu, K.: Time Series-Based Malware Detection Using Hardware Performance Counters. In: 2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST). pp. 102–112. IEEE, Tysons Corner, VA, USA (Dec 2021). https://doi.org/10.1109/HOST49136.2021.9702291, https://ieeexplore.ieee.org/document/9702291/
20. Kuruvila, A.P., Meng, X., Kundu, S., Pandey, G., Basu, K.: Explainable Machine Learning for Intrusion Detection via Hardware Performance Counters. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **41**(11), 4952–4964 (Nov 2022). https://doi.org/10.1109/TCAD.2022.3149745, https://ieeexplore.ieee.org/document/9706299/
21. Mushtaq, M., Akram, A., Bhatti, M.K., Chaudhry, M., Lapotre, V., Gogniat, G.: Nights-watch: A cache-based side-channel intrusion detector using hardware performance counters. In: Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy. pp. 1–8 (2018)
22. Nascimento, P.P.d., Pereira, P., Mialaret, J.M., Ferreira, I., Maciel, P.: A methodology for selecting hardware performance counters for supporting non-intrusive diagnostic of flood DDoS attacks on web servers. Computers & Security **110**, 102434 (Nov 2021). https://doi.org/10.1016/j.cose.2021.102434, https://linkinghub.elsevier.com/retrieve/pii/S0167404821002583
23. Olani, G., Wu, C.F., Chang, Y.H., Shih, W.K.: DeepWare: Imaging Performance Counters with Deep Learning to Detect Ransomware. IEEE Transactions on Computers pp. 1–1 (2022). https://doi.org/10.1109/TC.2022.3173149, https://ieeexplore.ieee.org/document/9770351/
24. Omotosho, A., Welearegai, G.B., Hammer, C.: Detecting return-oriented programming on firmware-only embedded devices using hardware performance counters. In: Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing. pp. 510–519. ACM, Virtual Event (Apr 2022). https://doi.org/10.1145/3477314.3507108, https://dl.acm.org/doi/10.1145/3477314.3507108
25. Oshana, R., Thornton, M.A., Larson, E.C., Roumegue, X.: Real-Time Edge Processing Detection of Malicious Attacks Using Machine Learning and Processor Core Events. In: 2021 IEEE International Systems Conference (SysCon). pp. 1–8. IEEE, Vancouver, BC, Canada (Apr 2021). https://doi.org/10.1109/SysCon48628.2021.9447078, https://ieeexplore.ieee.org/document/9447078/

26. Polychronou, N.F., Thevenon, P.H., Puys, M., Beroulle, V.: A hybrid solution for constrained devices to detect microarchitectural attacks. In: 2023 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW). pp. 259–269 (2023). https://doi.org/10.1109/EuroSPW59978.2023.00033
27. Pundir, N., Tehranipoor, M., Rahman, F.: RanStop: A Hardware-assisted Runtime Crypto-Ransomware Detection Technique (Nov 2020). https://doi.org/10.48550/arXiv.2011.12248, http://arxiv.org/abs/2011.12248, arXiv:2011.12248 [cs]
28. Satilmiş, H., Akleylek, S., Tok, Z.Y.: A systematic literature review on host-based intrusion detection systems. IEEE Access **12**, 27237–27266 (2024). https://doi.org/10.1109/ACCESS.2024.3367004
29. Sayadi, H., Gao, Y., Mohammadi Makrani, H., Lin, J., Costa, P.C., Rafatirad, S., Homayoun, H.: Towards Accurate Run-Time Hardware-Assisted Stealthy Malware Detection: A Lightweight, yet Effective Time Series CNN-Based Approach. Cryptography **5**(4), 28 (Oct 2021). https://doi.org/10.3390/cryptography5040028, https://www.mdpi.com/2410-387X/5/4/28
30. Singh, B., Evtyushkin, D., Elwell, J., Riley, R., Cervesato, I.: On the Detection of Kernel-Level Rootkits Using Hardware Performance Counters. In: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security. pp. 483–493. ACM, Abu Dhabi United Arab Emirates (Apr 2017). https://doi.org/10.1145/3052973.3052999, https://dl.acm.org/doi/10.1145/3052973.3052999
31. Singh, Y., Kuruvila, A.P., Basu, K.: Hardware-assisted Detection of Malware in Automotive-Based Systems. In: 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE). pp. 1763–1768. IEEE, Grenoble, France (Feb 2021). https://doi.org/10.23919/DATE51398.2021.9474053, https://ieeexplore.ieee.org/document/9474053/
32. Thevenon, P., Riou, S., Tran, D., Puys, M., Polychronou, N.F., El-Majihi, M., Sivelle, C.: iMRC: Integrated monitoring & recovery component, a solution to guarantee the security of embedded systems. J. Internet Serv. Inf. Secur. **12**(2), 70–94 (2022). https://doi.org/10.22667/JISIS.2022.05.31.070, https://doi.org/10.22667/JISIS.2022.05.31.070
33. Wang, H., Sayadi, H., Pudukotai Dinakarrao, S.M., Sasan, A., Rafatirad, S., Homayoun, H.: Enabling Micro AI for Securing Edge Devices at Hardware Level. IEEE Journal on Emerging and Selected Topics in Circuits and Systems **11**(4), 803–815 (Dec 2021). https://doi.org/10.1109/JETCAS.2021.3126816, https://ieeexplore.ieee.org/document/9610047/
34. Wang, X., Karri, R.: Numchecker: detecting kernel control-flow modifying rootkits by using hardware performance counters. In: Proceedings of the 50th Annual Design Automation Conference. DAC '13, Association for Computing Machinery, New York, NY, USA (2013). https://doi.org/10.1145/2463209.2488831, https://doi.org/10.1145/2463209.2488831
35. Wang, X., Karri, R.: Reusing Hardware Performance Counters to Detect and Identify Kernel Control-Flow Modifying Rootkits. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **35**(3), 485–498 (Mar 2016). https://doi.org/10.1109/TCAD.2015.2474374, http://ieeexplore.ieee.org/document/7229276/
36. Woralert, C., Liu, C., Blasingame, Z.: HARD-Lite: A Lightweight Hardware Anomaly Realtime Detection Framework Targeting Ransomware. IEEE Transactions on Circuits and Systems I: Regular Papers **70**(12), 5036–

5047 (Dec 2023). https://doi.org/10.1109/TCSI.2023.3299532, https://ieeexplore.ieee.org/document/10208245/

37. Woralert, C., Liu, C., Blasingame, Z., Yang, Z.: A Comparison of One-class and Two-class Models for Ransomware Detection via Low-level Hardware Information. In: 2023 Asian Hardware Oriented Security and Trust Symposium (AsianHOST). pp. 1–6. IEEE, Tianjin, China (Dec 2023). https://doi.org/10.1109/AsianHOST59942.2023.10409333, https://ieeexplore.ieee.org/document/10409333/
38. Zhou, H., Wu, X., Shi, W., Yuan, J., Liang, B.: HDROP: Detecting ROP Attacks Using Performance Monitoring Counters. In: Information Security Practice and Experience, vol. 8434, pp. 172–186. Springer International Publishing, Cham (2014). https://doi.org/10.1007/978-3-319-06320-1_14, http://link.springer.com/10.1007/978-3-319-06320-1_14, series Title: Lecture Notes in Computer Science