iMRC: Integrated Monitoring & Recovery Component, a Solution to Guarantee the Security of Embedded Systems

Pierre-Henri Thevenon[1]*, Sébastien Riou[2], Duc-Minh Tran[3], Maxime Puys[1], Nikolaos Foivos Polychronou[1], Mustapha El Majihi[1], and Camille Sivelle[1]

[1]Univ. Grenoble Alpes, CEA, LETI, DSYS, F-38000, Grenoble, France
{Pierre-Henri.Thevenon, Maxime.Puys, Nikolaos.Polychronou, Mustapha.Elmajihi, Camille Sivelle}@cea.fr

[2]Tiempo Secure, Montbonnot-Saint-Martin, F-38330, France
Sebastien.Riou@tiempo-secure.com

[3]Université Paris-Saclay, CEA, LIST, Palaiseau, F-91120, France
Duc-Minh.Tran@cea.fr

Received: March 25, 2022; Accepted: May 6, 2022; Published: May 31, 2022

Abstract

In recent years, the security of connected objects has become a real challenge. Indeed, more and more IoT devices are being built for increasingly critical applications and as shown by multiple famous botnet attacks such as Mirai, IoT devices are often poorly protected. In this paper, we introduce a new solution called iMRC (integrated Monitoring & Recovery Component) to improve the resilience of embedded systems in case of proven attacks. This innovative solution integrates a hardware component whose main function is to extract the hardware performance counters of the processor in order to be analyzed by the artificial intelligence of the control server. This one is able to remotely restore the devices to a known secure state upon detection of malwares or other abnormal behaviors. We define a use case based on a home automation network in which the iMRC component is added to a gateway. We implement a set of scripts reproducing malicious behaviors in order to test our detection capabilities and show that all malwares are detected within less than 20 seconds after the launch of a malware execution.

Keywords: IoT, IIoT, cybersecurity, resilience, AI, detection, secure element

1 Introduction

The economic stakes of the security of connected objects are considerable. Indeed, the global internet of things (IoT) market is projected to grow from USD 478.36 billion in 2022 to USD 2,465.26 billion by 2029[1]. Yet, recent years have shown a variety of cyberattacks caused by malwares able to spread widely (Stuxnet [1], Mirai [2], BlackEnergy [3], or BASHLITE [4]). This leads to the large-scale deployment of communicating objects in our daily life systems and industrial infrastructures, including the most critical ones, to be accompanied by proven risks. Moreover, on the 7th of June 2019, the Cybersecurity Act has

Journal of Internet Services and Information Security (JISIS), volume: 12, number: 2 (May), pp. 70-94
DOI:*10.22667/JISIS.2022.05.31.070*

*Corresponding author: CEA-Leti, 17 rue des Martyrs, 38054 GRENOBLE CEDEX 9, France. This work has been partially funded by the French *Fonds pour l'Innovation et l'Industrie (FII)*'s *Grand Défi Cyber*.

[1]https://www.fortunebusinessinsights.com/industry-reports/internet-of-things-iot-market-100307

officially been adopted by the European Union and will allow the definition of a European cybersecurity certification framework. Such certification will become mandatory to sell in Europe and many other countries are following similar trends. This challenging context requires improving the resilience of connected objects. First, since most IoT attacks are often related to malware infections, IoT devices should either be resilient to such attacks or allow to detect them before they can spread. Additionally, IoT devices are regularly the target of innovative attacks and are specifically hard to patch given their location on the field, leading to a need to detect future threats. Finally, comes the question of how to take back control of the device once affected by a malware. Such a situation currently requires manual intervention, on the field and device per device, which is peculiarly tedious when dealing with huge botnet attacks. While solutions exist to improve the security of embedded systems, it is not possible to guarantee the security of an IoT or I-IoT device on a perennial basis. Only a solution based on the use of a secure hardware element with a minimalist architecture, an independent vulnerability assessment methodology and a self-defense capability can ensure maximum security.

Related Works: Many ambitious projects involve the deployment of a large fleet of connected embedded systems. Developers of these solutions are constantly pushed to design increasingly complex systems while meeting tight time-to-market targets. This limits the possibility of proper threat assessment and it is highly likely that vulnerabilities will remain unknown before deployment in the field. In theory, current solutions such as ARM TrustZone allow manufacturers of embedded systems or SoCs to design systems with certain security attributes [5]. In practice, many SoCs designed in this way have proven to be vulnerable because the designers of these SoCs inherited the complex task of properly integrating the security solution into the SoC architecture. Even when the hardware is properly secured, a single software flaw can still compromise the security of the entire system. For instance, one can cite the attack extracting Bitlocker keys from TPM [6], or LVI [7] vulnerability against Intel SGX. To solve such complexity, manufacturers of SoCs for the Internet of Things (IoT) have recently begun to design integrated Secure Elements (iSE) into their processors to manage certain security functions [8]. iSEs are the equivalent of a discrete smart card embedded in the heart of an SoC. The main advantage over solutions such as TrustZone is that the SoC manufacturer cannot compromise the security of the iSE if mistakes are made during integration. These products are subject to the same independent vulnerability assessment process as the smart card. The solution is therefore likely to achieve the highest level of security. Examples of existing iSE are Tiempo Secure's TESIC [9] or Secure-IC's Securyzr [10].

Anomaly detection with machine learning and deep learning has been used for network intrusion detection in many contexts. Leung and Leckie showed an application of this technique with a clustering algorithm [11], Koc et al. [12] used Hidden Naive Bayes multi classifier to detect attacks in the KDD 1999 data set. More recent deep learning techniques have also been applied. Khan et al. introduced a new intrusion detection method based on convolutional neural network by using network spectrogram images generated using Fourier transform [13]. Ullah et al. proposed a solution with 1D, 2D and 3D convolutional neural networks combined with transfer learning [14]. Yazdinejadna et al. presented an approach SDN-based architecture intrusion detection system for attack detection and malicious behaviors in the data plane [15]. Vinayakumar et al. introduced a highly scalable and hybrid DNNs framework which can be used in real-time network traffic monitoring and host-level events [16].

Most recent studies for malware detection focus on identifying malicious applications on Android operating system by analyzing various aspects. Yousefi-Azar et al. proposed a solution with neural network by extracting static features of binary files [17]. Kim et al. presented a multimodal deep learning approach using static features by analyzing manifest, dex and APK files. [18]. Karbab et al. proposed a sequences mining with neural networks that takes the raw sequences of API method calls[19]. Ma et al. introduced an ensemble learning approach by analyzing API calls, API frequency and API sequence

aspects[20]. Mercaldo and Santone presented a method to discriminate between malicious and legitimate applications in Android and iOS by representing them as grey-scale images and classifying them with deep neural network[21].

However, while the iSE approach solves the security vs. complexity problem, the main system can still be infected by malwares. To tackle this problem, an impressive amount of research works have proposed various solutions on host-based malware detection mechanisms. First can be cited various operating-system and kernel specific solutions, often relying on syscall monitoring, alongside memory checks to detect ROP based exploits [22, 23, 24, 25]. Various commercial solutions work with similar ideas [26, 27, 28, 29, 30, 31, 32, 33]. Many other related works rely on Hardware Performance Counters (HPC) to detect malicious behavior. In 2019, Das et al. proposed a survey on the use of HPC for security [34]. They classify existing works based on their determinism and their measurement error handling. Among other security-related use cases for HPC, we hereafter showcase specifically malware detection solutions. In 2013, Demme et al. introduced the idea of performing supervised machine learning on HPC in order to detect malwares [35]. Also in 2013, Wang and Karri proposed Numchecker, a virtual machine monitor able to detect control-flow modifying kernel rootkits in a guest virtual machine [36]. In 2014, Tang et al. extended [35] with the use of unsupervised machine learning [37], followed by Garcia-Serrano in 2015 [38]. Also in 2014, Zhou et al. relied on the branch mis-prediction performance counter in order to detect ROP attacks [39]. Similar works were proposed in 2014 by Bahador et al. [40], in 2016 by Wang and Backer [41] and in 2018 by Das et al. [42]. In 2016, Wang et al. present a software based online malware detection relying on HPC signatures [43]. Similar results are proposed by Peng et al. in 2016 [44]. Also in 2016, Jyothi et al. leverage HPC to detect DoS attacks [45] and Chiappetta et al. proposed correlation, anomaly detection and neural-network based approaches to detect Cache Side-Channel attacks in real time [46]. They are followed in 2017 by Zhang et al. who built on the work of [36] to detect DoS attacks in IaaS infrastructures such as Amazon's EC2 [47]. In 2017, Singh et al. built on previous works to detect malwares using HPC but specifically focus on kernel-level rootkits [48]. Also in 2017, Patel et al. introduced an FPGA implementation of offline AI classifiers for malware detection using HPC data sets. However, they cannot perform online monitoring. Most of attacks targeting the hardware vulnerabilities such as cache attacks or Rowhammer found in most of processor architecture can also be detected using HPC registers as it was shown in a survey written by authors in 2021 [49]. In 2019, Li et al. proposed a detection solution to detect Rowhammer and Spectre attacks with various classifiers, such as Logistic Regression, Support Vector Machine and ANN [50]. Also in 2019, Gulmezolu et al. develop unsupervised Recurrent Neural Networks-based models to learn the normal behavior of the system, in order to detect Meltdown, Spectre, Rowhammer and Zombieload attacks on Intel processors without including them in the training task [51]. Finally, in previous work, authors proposed in 2021 a Logistic Regression classifier has also been used to detect a larger set of SATHV on a ARM7 processor [52]

However, all of these solutions are for the most software pieces (kernel modules, executables, etc), part of the execution environment. This implies both that they can be vulnerable to attacks themselves as other applications, but also that they risk disrupting the normal application flow with their own execution, thus reducing the detection accuracy. Other part of aforementioned works perform offline detection on data sets, rather than online detection within the system. To the best of our knowledge, the only existing hardware implementations of AI based online malware detectors was proposed by Das et al. in 2015 [53]. They designed an FPGA implementation of an AI based malware detector relying on system call patterns.

Finally, even if iSE help reducing attack surface and HPC based monitoring allows to detect attacks, they still require the infected device to be manually recovered. Worse, there is no way for the object manager to know which objects are affected and to regain full control of them. A classical solution involves *image swapping*, meaning a way to reboot the device on another system image (for instance on a degraded mode in case of issue). This can either be done with physical connection to the device or

remotely with an API. MCUboot [54] is clearly becoming the leading solution for image swapping. A more versatile solution consists in *checkpoints*, that is, saving the internal state of the system at a given time and being able to restore it later rather than booting on a different image. Obviously, many questions are raised regarding such solutions (e.g., time intervals, checkpoint storage, incrementality, checkpoint contents, etc). Many solutions have been proposed in the past 25 years [55, 56, 57, 58, 59, 60, 61]. However, this raises the question of the security of such remote API, leading either the device to not be recoverable or to be switchable by an untrusted party. Such remote image swapping API can be more resilient if implemented in hardware as it becomes less subject to a malware running on the IoT device. An even solution could be to rely on a secure element to control the image swapping.

Contributions: In this paper, we introduce the concept of *integrated Monitoring & Recovery Component* (iMRC). Such a solution is able to address the problem of maintaining a large fleet of IoT devices secure. We propose a new architecture in which the iSE can take control over the rest of the SoC and the non-volatile memory. On the cloud side, a monitoring server is dedicated to the detection of malicious behaviors using an Artificial Intelligence (AI) system but also to the management of updates and the response to security incidents. More formally, the iMRC solution allows to:

- Secure the boot procedure to ensure the integrity of firmware;
- Communicate with a remote control server to notify the status of each device;
- Locally detect a loss of communication with the control server;
- Remotely detect an abnormal behavior using artificial intelligence on each connected nodes;
- Update the whole fleet of the connected devices after the detection and analysis of a vulnerability in a device.

We define a use case based on a home automation network in which the iMRC component is added to a gateway. We implement a set of scripts reproducing malicious behaviors in order to test our detection capabilities and show that all malwares are detected within less than 20 seconds after the launch of a malware execution.

Outline: Section 2 will describe the iMRC component and its behavior. Section 3 will present more in depth the inner-workings and implementation choices. Then Section 4 summarizes our test malwares and the detection results from the cloud AI. Finally, we conclude in Section 5.

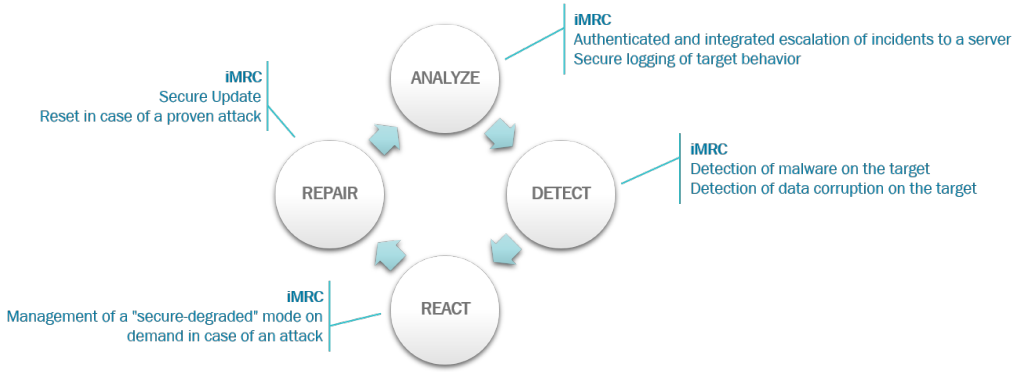

Figure 1: iMRC's Goals

2 The iMRC Solution

In this section, we first give a general description of the iMRC solution. More technical information related to the implementation will be presented in Sections 3 and 4. Section 2.1 will first introduce the core concepts of the iMRC SoC. Section 2.2 will then describe the control server. Section 2.3 will give insights on how we use Artificial Intelligence (AI) in order to detect malwares. Finally, Section 2.4 and 2.5 will respectively describe the operating modes the threat model we considered. The missions of this solution are partly based on the five priority functions defined by the NIST cybersecurity framework to achieve a secure system [62]. Figure 1 shows the detection, response, repair, and analysis functionalities of a connected object to resist attacks and how the iMRC solution can help.

As described in Figure 2, the solution is based on a certified secure element that allows to regain control of the connected object in case of detection of a malicious behavior. This secure element is based on a monitoring solution that extracts internal signals from the processor and the SoC. This data is then sent to a secure server and analyzed using artificial intelligence algorithms to detect the presence of malware or any other attack that modifies the behavior of the connected object. A secure communication channel between the secure element and the server allows to know in real time the status of each connected object and to apply updates to a network of vulnerable devices as soon as an attack is detected.

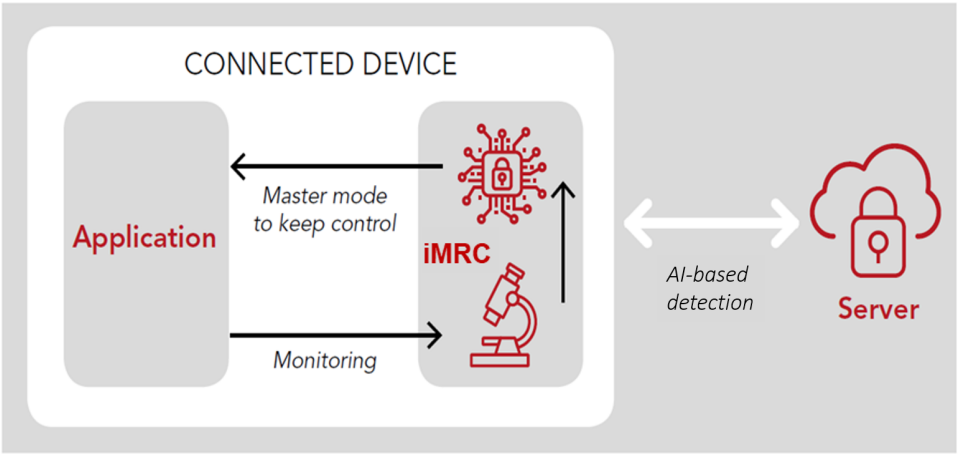

Figure 2: Presentation of the iMRC Solution

The control server therefore receives a report on the internal state of each object at regular intervals. This allows to provide data on the object's network activities, the memory access activity (volatile or not), the computation load, etc. to the artificial intelligence system. This AI system trained with this data will be able to identify the presence of malicious code even if it is only waiting for its activation. It is similar to a HIDS. Moreover, this large-scale observability, i.e., on the scale of a fleet of connected objects, should make it possible to optimize and improve the solution's detection capabilities. In the event of a proven attack, the control server can request the activation of a secure degraded mode that ensures only critical functionalities until a patch for the vulnerability is available. The iSE may fail to establish a connection with the control server for a variety of reasons (accidents or attacks). After a predefined time, the iSE can also take the initiative to activate the secure mode to be able to take control of the SoC. Once the connection with the control server is re-established, the iSE can send a report on the incident and wait for instructions from the control server.

The iMRC offers a lightweight auditing module that can identify complex attack behaviors and react accordingly, depending on the desired security level. The advances are twofold. First, iMRC uses machine learning techniques to automatically build an audit module, starting with the expertise of point

attackers. Second, the module is programmable, and thus flexible enough to be deployed in multiple IoT domains. In the case of a non-critical system, the control recovery process can be simple and generic. In the case of a critical system (e.g., industrial systems), the control recovery process can take into account system-specific constraints to maintain certain services throughout the process.

2.1 Description of the System-on-Chip

We first describe the innovative System-on-Chip (SoC) used as part of the iMRC solution. Figure 3 describes the differences between a classic SoC and the iMRC enhanced SoC.

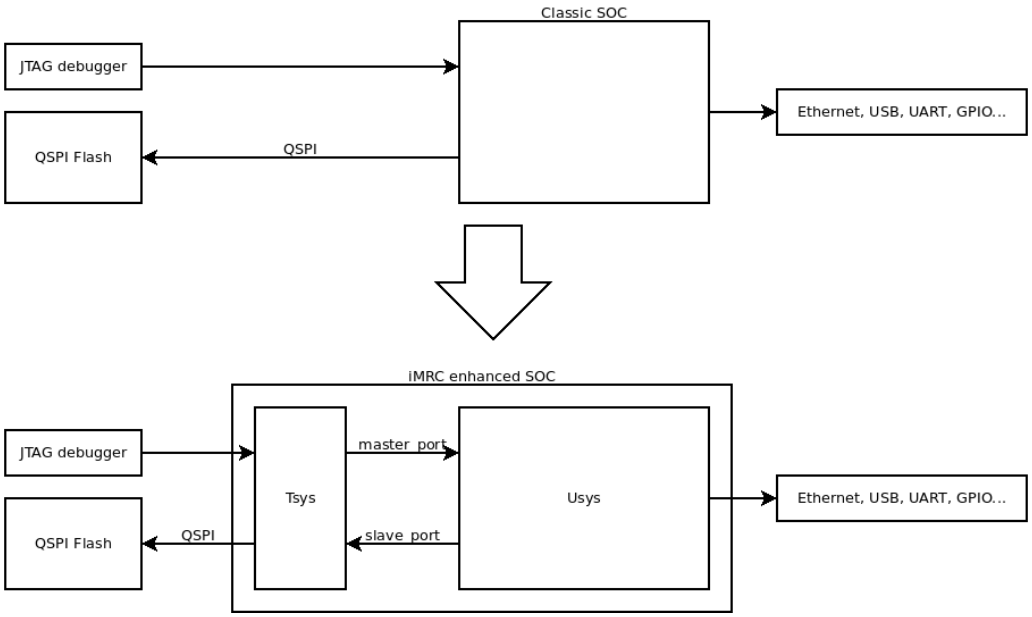

Figure 3: System-on-Chip's Architecture

In contrast with a classic monolithic SoC that only contains the applicative processor, the System-on-Chip of the iMRC solution is divided into three parts:

- An applicative part called Usys;
- A secured part called Tsys;
- A measurement module.

Usys – Applicative World: The application part called Usys for *Untrusted System*, contains all the functionalities of a classic applicative processor. It includes at least:

- A Central Processing Unit (CPU);
- A RAM memory;
- A communication interface to access the network;
- A communication interface with Tsys.

No internal non-volatile memory is necessary, the CPU starts directly on the RAM memory (which will be loaded beforehand by Tsys via an interface called *master_port*). As this applicative part can be complex, we consider as a typical case a multi-core architecture running Linux (e.g., the processor of a Raspberry Pi).

Tsys – Secure world: The secured part of the Soc called Tsys for *Trusted System*, can be considered as a secure element integrated in the SoC (an iSE), able to take the control of the complete SoC in order to recover the system after a malware detection or in case of a lost connection with the control server. Tsys ensures all the features offered by a classic iSE, including:

- Confidentiality and integrity of the iMRC firmware and data (even if stored in an external memory), with a very high level of protection (Common Criteria EAL5+ certified) against all types of attacks, including physical attacks (e.g., side-channel or fault injection) aimed at recovering the encryption keys stored in the iMRC,
- Confidentiality and integrity of arbitrary data coming from the application,

Tsys also manages the secure boot of the file system (call *rich system*) and secure update.

Measurement module: The measurement module is used to extract signals from the processor and peripherals in order to gather information about the running processes. This allows for malwares or malicious behavior detection. The signals used to detect an abnormal behavior are HPC (Hardware Performance Counter) registers that gave metrics about the use of main components of the SoC (Memory, bus, etc.). The measurement module will also optimize data sent to the server by filtering all the extracted signals.

In this solution, the flash memory is divided into several partitions to store the data and the software of Usys and Tsys as well as the recovery software of Usys. All the flash memory is encrypted to avoid leaking potentially confidential data (e.g., configuration).

2.2 Control Server

The main objective of the control server is to manage all the IoT nodes integrating iMRC solution and transfer all data from the measurement module to the artificial intelligence to detect abnormal behaviors on a node. As shown in Figure 4, the control server has two interfaces: one for users and the second for connected objects.

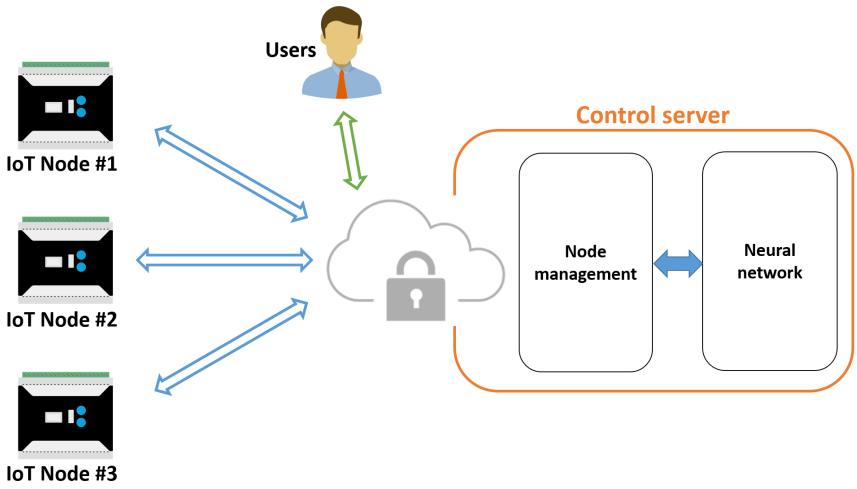

Figure 4: Control Server's Architecture

On the user interface, exchanges are protected by the standard TLS protocol. The server authenticates users using their x509 certificates. The control server offers a fine-grained access-control policy on the functionalities it provides. For instance, the operator of the fleet of connected objects could completely manage all the IoT nodes (add or remove an IoT node) while a developer of the embedded software could upload an application update package or connect a debugger to an object on which a problem has been detected. Debugging on a connected object is the only service that requires the simultaneous connection of the user and the connected object concerned. For the other services, the control server relies on a database to keep track of the last status of each object and to know which commands to send at the next connection.

On the interface of the connected objects, the control server treats each connection according to the status associated with the object at the origin of the connection. The communication between Tsys and the control server is protected by a secure protocol of type SCP03, lighter than TLS. In nominal operation, the control server answers to the connection of an object by a request for transfer of the data of the measurement block. The server then uses this data to feed the AI-based malware detection system described in the next section.

2.3 AI-based Malware Detection

Most malware detection solutions in the industry are based on a basic method: signature-based detection (e.g., Splunk, McAfee, SolarWinds), in which the monitoring system compares its behavior with patterns defined in the malware knowledge database [63]. This solution allows to detect them efficiently and quickly without having positive false but has some limitations. Indeed, due to the lack of malware signatures for IoT devices, the signature-based system is not able to detect unknown or zero-day malware. To overcome this difficulty, an anomaly-based detection solution can close this security gap by finding both known and unknown attacks. This technique uses an unsupervised learning AI model that understands the normal behavior of the system after training. The choice of unsupervised algorithms is more suitable for industrial needs because it is no longer necessary to label the data, which is an extremely costly task. Our solution for the detection of new applicative malwares is inspired by our previous work on the network intrusion detection with unsupervised anomaly detection techniques [64], in which we succeeded to detect 7/11 attacks on CICIDS2017 data set without any false positives. In this work, instead of using metadata from network packets, we use HPC counters' values collected directly from the nodes. We train neural network based models such as Multilayer perceptron (MLP), Autoencoder (AE) and Bidirectional Long short-term memory (LSTM) that understand the behavior of benign applications and therefore can detect malicious behaviors from our malwares. The proposed solution is divided into 3 steps:

- **Training:** this step is used to train an AI model on the data during the period when only normal applications are executed. After this step, the model is able to build or predict the normal behavior of the system. An anomaly score is also computed to estimate the difference between the output constructed by the AI model and the expected output.

- **Calibration:** the goal of this step is to find the threshold S that allows to determine if a sample represents an abnormal behavior. To do this, the trained model is tested on a portion of data that comes only from normal applications, and the anomaly scores of this portion are calculated. The threshold S is determined by calculating the 99.9th percentile of these scores.

- **Detection:** At this point, the AI model takes the incoming data, produces the output and then computes the anomaly scores. These scores are compared with the threshold S obtained from the calibration. An anomaly score higher than the threshold translates into an anomaly and thus a

potential alert. The aggregation of alerts is implemented at the end to reduce false positives by ignoring anomalies that are isolated among normal behaviors. Only a group of anomalies (i.e., a group of adjacent samples with scores $>$ S) is considered true, and an alert will be sent to the Control Server.

2.4 Operating Modes

Depending on the context, the iMRC device will operate in one of the three following modes:

Nominal mode: Under normal circumstances, Tsys periodically retrieves data from the measurement block and attempts to connect to the control server at regular intervals to communicate the data. If the measurement data do not reveal anything alarming and the communication with the control server was successful, Tsys goes back to standby until the next analysis.

Malware detection: A malware is detected by Tsys when the control server's artificial intelligence processes the data from the measurement block, detects a malicious behavior and notifies Tsys. To detect Denial of Services attack against the communication channel between Tsys and the control server, Tsys can verify the connection status with the control server and analyze multiple failed connections to the control server as a malicious behavior without the help of the AI. If a malicious behavior is detected or if an explicit request comes from the control server, Tsys triggers the system control recovery phase.

Recovery mode: During the recovery phase, Tsys takes the control of Usys and stop the execution of all processes on it, it loads the recovery image into the Usys RAM and check its validity. After this procedure, Usys restarts on the recovery image and communication between Tsys and the control server is recovered. Tsys can then receive commands from the control server to analyze the details of the attack. In particular, one of the possibilities is to read memory areas to locate the malware and analyze its code. This is possible because the recovery image occupies only a tiny part of Usys' RAM and therefore leaves Usys' RAM largely untouched. In practice, there are physical means to ensure that the malware does not end up in the memory footprint of the recovery image but that is beyond the scope of this article. Finally, the control server can push an update to the whole fleet of embedded vulnerable devices which corrects the vulnerability.

2.5 Threat Model

This section gives insights on the cybersecurity risk analysis performed to validate the security of the iMRC solution. Even if the proposed solution can be used in most of embedded devices, a generic use case was defined around a critical industrial device to identify the main threats. This used case is mainly based on an industrial gateway that makes the bridge between the industrial world and the cloud world.

Figure 5 depicts our chosen use case. This industrial gateway is located in an electrical panel only accessible by authorized operators. This kind of gateway can be used in supermarket to manage the lights, power or refrigeration unit to keep cool fresh products. In such a scenario, the essentials assets to protect is the food and the technical operator want to be sure that he will be alerted in case of an alarm. In the industrial world, the gateway interacts with industrial sensors and actuators through various communication interfaces. Most of these wired interfaces are not secured because they rely on legacy communication protocols such as Modbus (RTU or TCP), Profibus, CAN, etc. The wireless interfaces are more recent and are often protected, even if it can exist some security vulnerabilities. In the cloud world, the gateway communicates with the SCADA through OPC-UA or Modbus TCP. It also offers services from a web server accessible from a smartphone and computer using BLE or Ethernet.

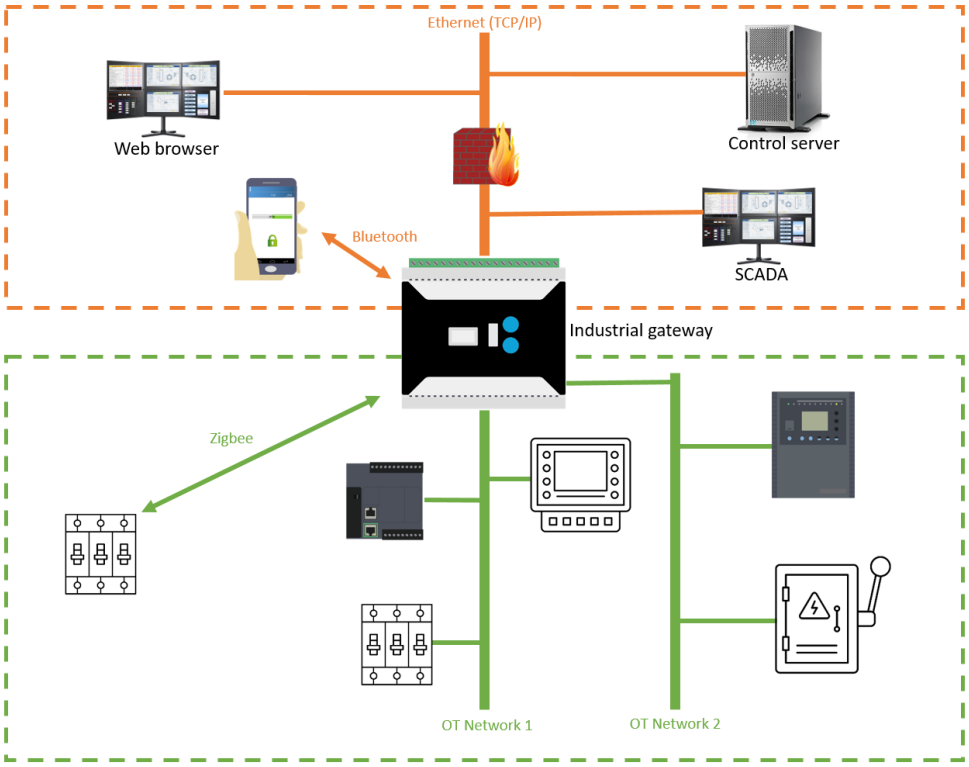

Figure 5: Use case Description

An industrial gateway is most often located in an electrical cabinet, which limits physical access to the device from a security point of view. Even if attacks requiring physical access to the device are less likely, they can still represent a significant risk that must be taken into account. Indeed, a maintenance technician or a corrupted technical operator can access or give access to the target device. Relatively simple attacks can then allow access to the debug/programming interface to extract the binary image of the embedded software. Thus, an attacker can analyze the binary to identify vulnerabilities.

The most likely attacks are those on the device's communication interfaces. The SHINE [65] and Ruggedtrax [66] studies, conducted in 2014 and 2015 respectively and based on the Shodan tool have shown that industrial devices are easily accessible from the Internet. From known or so-called zero-day vulnerabilities, it is possible to take control of a device or an entire network of devices in order to conduct larger-scale attacks. In the case of an industrial system, such an attack can have serious consequences with financial and ecological risks, but also potential human losses.

In addition to the threats concerning the use case, some threats are directly related to iMRC solution:

- Tsys secrets such as keys may be worth the effort of performing physical attacks such as side channel attacks, including advanced EM analysis, fault injection attacks, including multi-laser attacks, reverse engineering of ROM and part of glue logic or physical modification of the die using FIB. Advanced logical attacks are also relevant such as the analysis of scan chains and malicious test pattern injection, the analysis of internal architecture using debug or attacks from malicious code running on Usys.

- Usys is not supposed to manipulate high value assets. For this reason, physical attacks such as injection faults or side channel attacks are not considered as a realistic threat. Usys is mostly

concerned with remote attacks and trojans. Usys is allowed to have some weaknesses against those threats as long as the hardware guarantees that Tsys is able to take back the control and update the software if a vulnerability is discovered.

- The control server may be mainly targeted by remote attacks such as malformed inputs to gain execution privileges, queries from any computer or from a compromised node (Control server shall be safe even if Tsys is compromised on a node) or DDOS attacks.

3 Hardware Implementation of the System-on-Chip

This section describes more precisely how the iMRC solution has been implemented as a hardware component. In particular, Section 3.1 will describe the implementation of the internal secure element while Section 3.2 will present the implementation of the measurement module. The iMRC solution is based on SaxonSOC[2] which integrates two VexRiscv cores[3] written in an high-level hardware description language named spinalHDL. This open source SoC is optimized for FPGAs and is implemented on a Digilent Arty A7-100T evaluation board based on ARTIX-7 FPGA from Xilinx. The original SoC supports a CPU frequency of 100MHz and has an Ethernet interface. Figure 6 details the architecture of the iMRC system. Indeed, we find Usys and Tsys with the different interfaces between the two blocks and also with external components. The SaxonSoc has been adapted to be used for Usys. In this case, the measurement module has been integrated in the SoC in such a way that it has a physical access to the HPC registers implemented in the VexRiscv cores.

It has a physical access to the HPC registers implemented in the cores and it communicates with the Tsys block with an APB interface in a slave mode. Only Tsys is able to control Usys or to flash a program in its memory because it drives its Jtag and has a master access to Usys bus. On the other hand, Tsys integrates a secure element core with different units for memory management. A dedicated embedded

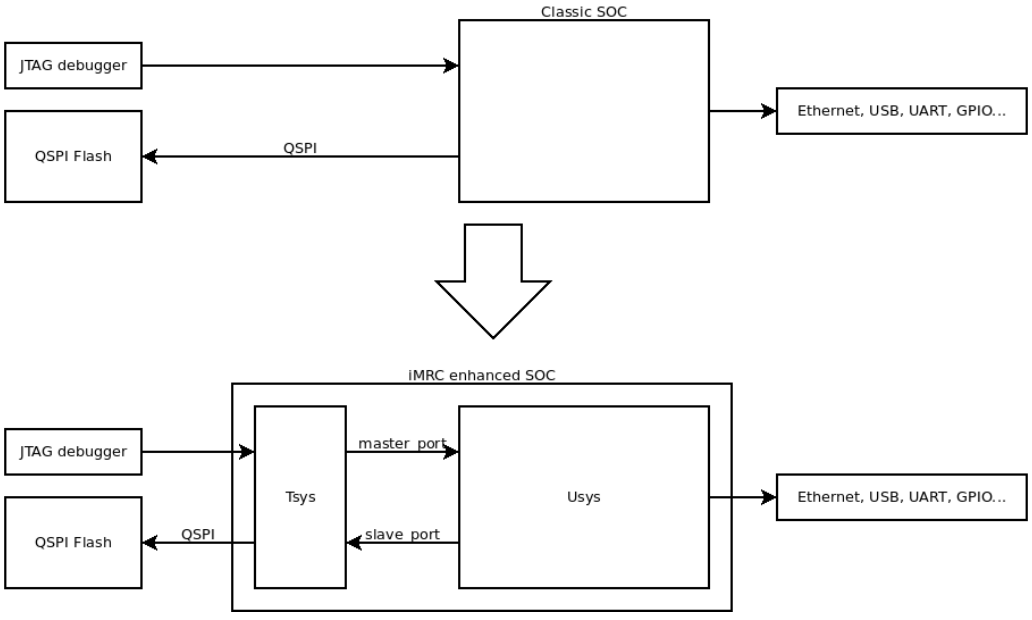

Figure 6: SoC architecture of iMRC

[2] https://github.com/SpinalHDL/SaxonSoc
[3] https://github.com/SpinalHDL/VexRiscv

Linux was integrated on these applicative processors of Usys using buildroot. It is used to execute all user application but also malicious programs.

3.1 Internal Secure Element – Tsys

Tsys is a secure element enriched in terms of its interfaces:

- A QSPI interface allows to control an external flash memory.
- A *master_port* interface allows Tsys to access the internal bus of Usys as a master.
- A *slave_port* interface allowing Usys to access the external flash memory through Tsys.

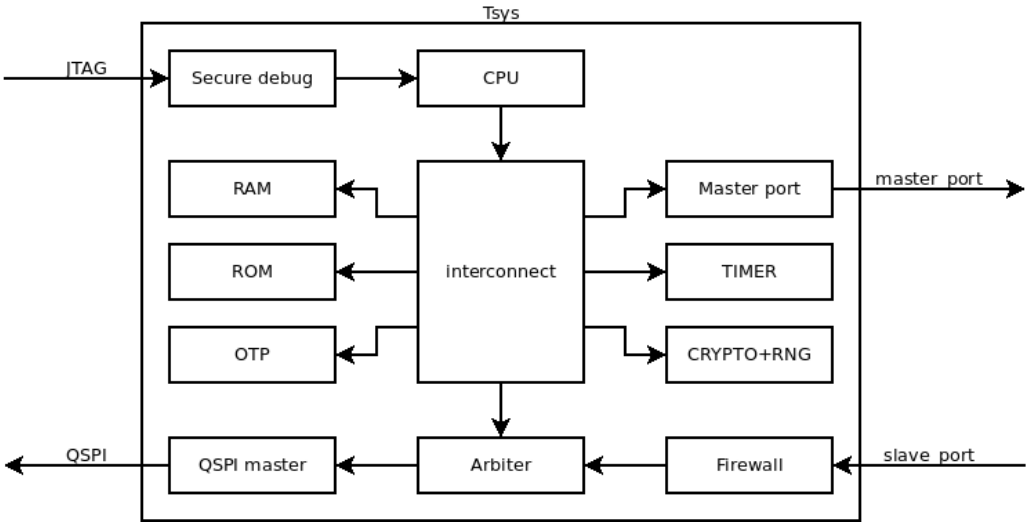

Figure 7: Minimal Secure World

As shown in Figure 7, the *master_port* interface, in addition to giving access to Usys bus, allows the following operations:

- Keep Usys in a reset state
- Keep all masters on Usys bus in a reset state (we call this state *halt*). When Usys is in the *halt* state, Tsys has exclusive access to all Usys resources.

The firewall allows Tsys to limit Usys' access to the external flash memory. Since the firewall is under the control of Tsys' CPU, the architecture ensures that Usys cannot corrupt the flash memory partitions that Tsys wishes to protect. It also ensures that Usys cannot monopolize access to the flash memory because Tsys can completely shut down the firewall to gain exclusive access.

3.2 HPC-based Measurement Module

The developed measurement system must be able to extract internal signals from registers called HPC (Hardware Performance Counters) available in most of processors (RISC-C ARM or Intel) in the RISC-V core, and then transmit them to Tsys. This hardware module must also be configurable from Tsys in order to select the relevant HPC registers and to configure the extraction frequency. In most of the state-of-the-art, the access to the HPC registers is normally done through the software layer because they are memory mapped by using a kernel module [34, 49]. In the proposed solution, this possibility is not used for two reasons:

- We need to access to the HPC registers through hardware layer in order to extract their values while executing programs. We don't have to halt cores to achieve this operation.
- Memory mapped HPC registers is a source of vulnerability. Some attacks are possible and they allow the attacker to corrupt their values.

The measurement module was developed as a hardware component which allows an access to the Hardware Performance Counters (HPCs) that are implemented, according to the RISC-V specification[4], in the VexRiscv cores in order to detect malware vulnerabilities. The measurement module is supposed to be a trusted module. It has different interfaces: an SPI interface with external world for debug and APB interface with Tsys for final integration. The image below describes an overview of the integration of this module in the SaxonSoc.

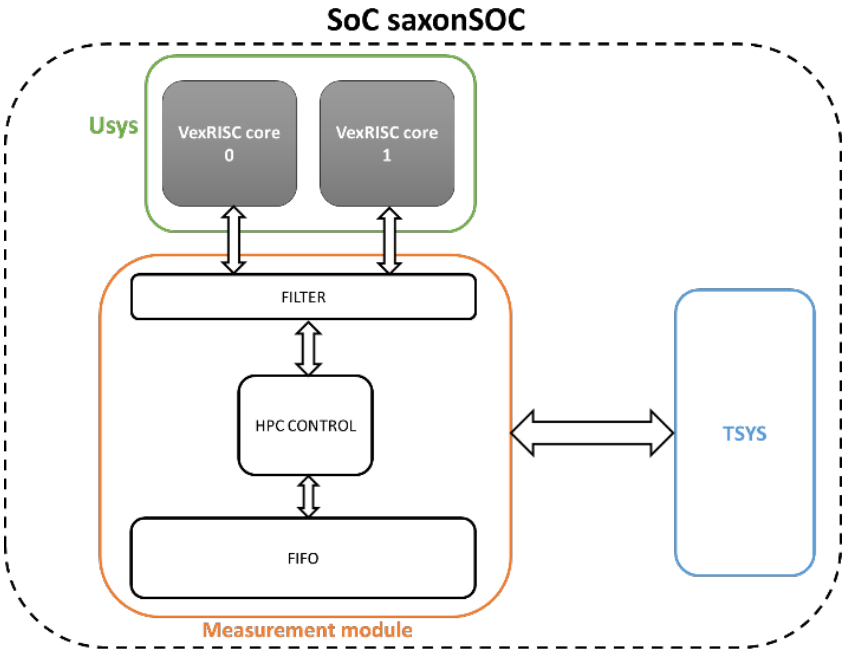

Figure 8: Integration of the Measurement Module in SaxonSoc

As shown in Figure 8, the values of the HPC registers are buffered in a FIFO memory then it could be transmitted through the interface selected by the user. As most of the HPC registers were not integrated in the VexRiscv core used for this implementation, we have developed and integrated the main interesting HPC registers. In order to minimize the amount of data transferred by Tsys to the control server, a compression filter was implemented to reduce the size of data to send by communication channels. It compresses each 32-bit HPC to an 8-bit value. This is done by some kind of normalization which can be described as $y = (x - a) >> b$ equation where x is initial 32-bit value, a and b are founded from 32-bit HPC values from execution statistics.

[4] https://github.com/riscv/riscv-isa-manual/releases/download/Priv-v1.12/riscv-privileged-20211203.pdf

4 Experimental Malware Detection

4.1 Methodology

In order to test our ability to detect malwares running on the Usys part of the iMRC, we relied on the same methodology than in [67] to extract the value of the HPC registers, validate, and optimize the machine learning models. As presented in Figure 9, the measurement module is implemented on a Digilent Arty A7-100T evaluation board based on ARTIX-7 FPGA from Xilinx (see Section 3). It is able to extract the value of different HPC registers from 2 applicative RISC-V cores on which an embedded Linux is running.

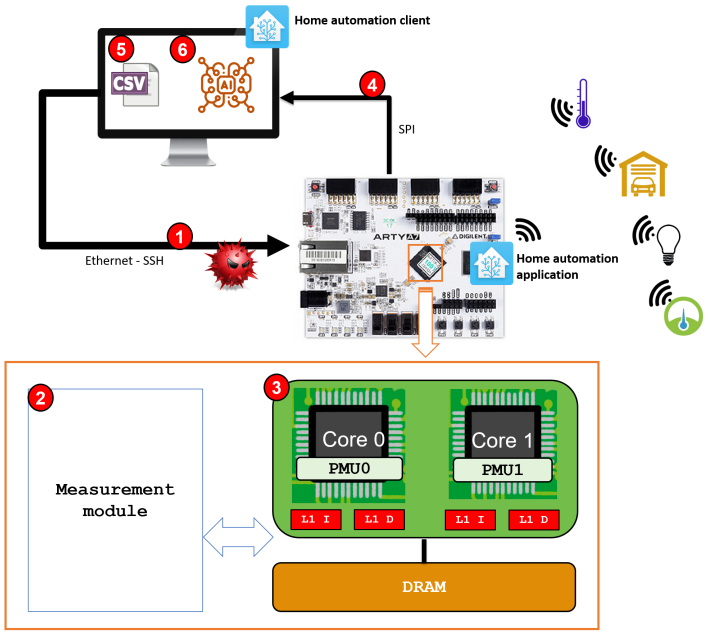

Figure 9: Testing Methodology

In steps (1) and (2) of this methodology, we configure which applications will be executed on the monitored RISC-V cores. An application can be either malicious or a legitimate user software. We also configure the measurement module by choosing the HPC registers we want to analyze, the periodicity of the measurements and the filtering options. During step (3), as applications run on the system, the measurement module periodically read HPC registers in each cores and store them in a FiFo. In step (4), the measurements saved in the FiFo of the measurement module are extracted by a computer through a SPI debug interface. Finally, in the last steps, the extracted raw data are converted to generate a labeled data file that is used to train and test AI models.

4.2 Datasets

A complete test based on a home automation use case integrating a wireless gateway and controlling several sensors/actuators (weather station, motion detector, light bulb, etc.) has been set up. This testbed integrates two different networks:

- The first network is the one between the home automation gateway and the IoT devices on WiFi

(weather station) and Z-Wave wireless communication interfaces (carbon dioxide detector, connected light bulb and socket, presence and door opening detector).

- The second network is the one between the home automation gateway, the user station allowing to control the home automation network (web interface of the Home Assistant software [5]) and the PC of a potential attacker. These devices communicate through an Ethernet interface.

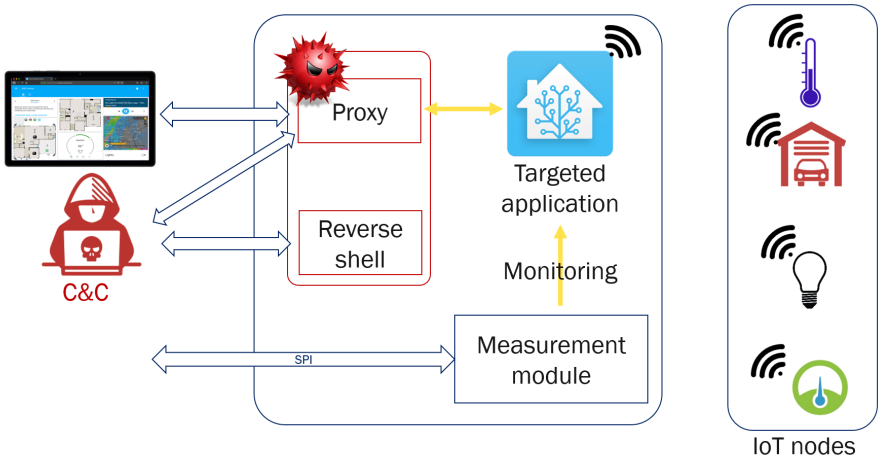

Figure 10: Home Automation Application and Malicious Softwares

Malicious software used to test the solution are not real malwares but have been developed to modify the behavior of the system like the final impact of the malwares. This allows us to precisely study how slight modifications on the behavior of the attacker can affect the detection. It is therefore considered that the attacker is already present inside our system and has elevated privileges on the system. The following application malwares have been used in the tests:

1. **Denial of service:** the main objective of this malicious software is to block all communication between the gateway and others IoT Devices or the Home automation client.
2. **Modification of application responses:** The attacker is able to corrupt the data sent by the home automation application to falsify the status of connected objects for example.
3. **Data-logger 1:** This malicious software sends a large volume of information to the attacker's website (spyware, memory dump)
4. **Data-logger 2:** This malicious software sends information relative to the user's application to the attacker's website (spyware that retrieves data from the application and connected objects)
5. **Man-in-the-Middle, injection:** The attacker can modify the contents of the user application, the requests or can send false requests to the server (e.g. to turn off an electrical outlet)
6. **Reverse shell:** This malware authorizes the attacker to open a console on the system, waiting for a remote activation.
7. **Cryptolocker:** This malware is able to encrypt critical data on the disk. A lot of ransomwares use this kind of malicious behavior.

[5] https://www.home-assistant.io/

The malwares have been designed to be configurable, in particular to test the performance of our detection system by giving the malware a greater or lesser impact. For example, in the case of data-logger malware, it is possible to configure the volume of data to be sent or the frequency of sending data over the network. The training data set, contains about 150000 samples with HPC values exclusively collected from benign execution program. On the test data set, each malware is launched 10 times. The chosen HPC for this test are:

- **MEM_ACCESS:** Data memory access.
- **L1I_CACHE:** Level 1 instruction cache access.
- **L1D_TLB_REFILL_ST:** Level 1 data TLB refill - Write.
- **L1D_CACHE_LD:** Level 1 data cache access - Read.
- **BR_IMMED_SPEC:** Branch speculatively executed - Immediate branch.

4.3 Neural Networks Architecture

The goal of neural networks in our experiments is to learn the normal behavior of the system. There are several neural network architectures to solve this problem. In our experiment, the 2 methods used are: Reconstruction and Prediction.

Reconstruction: the main idea behind this method is to use an Autoencoder network to transform input values into a compressed vector representation, and then try to decode this vector in order to have the same values as the input vector. It assumes that the neural network will correctly encode and decode the subset of samples from legitimate behaviors, while it will not be able to do so on illegitimate ones. In this paper, we use a neural network whose number of neurons in each layer is as follows: (number of features)-32-16-8-16-32-(number of features). The activation function chosen is Rectified Linear Unit (ReLU) [68] to obtain a faster convergence and reduce the gradient disappearance. The mean square error function (i.e. MSE) [69] calculated between the output of neural networks and the input vector, is used to evaluate the performance during training and to calculate the anomaly score in calibration and detection. A drawback of this method is that it does not consider learning the context of surrounding samples, as we only take one sample as the only input of neural network. So, preceding values from HPC do not have any effect on the current values.

Prediction: the neural networks of this method take a sliding window of t consecutive samples and predict the next (t+1) sample. As with the Reconstruction method, these models are trained only with legitimate behavioral data and therefore able to predict the normal operation of the system. The activation function, the way to calculate the anomaly scores and the threshold are similar to the Reconstruction method. The main difference between this method and the first one is that the neural networks take into account the previous context. The networks used in this method are Multi-layer perceptron with a structure (number of features * window size)-128-64-32-16-(number of features) and Bidirectional Long short-term memory with 100 units. The sliding window size in the test is set to 10. Whereas MLP takes into account the context but not sequentially, LSTM can overcome this challenge as it fits the recurrent demand for the Prediction method.

For both methods, when these networks fail to predict an desired output sample, which means the anomaly score of this sample is higher than the threshold obtained from the calibration, this sample is classified as an anomaly.

4.4 Used Metrics

In this section we present metrics used for evaluation. First, the classic metrics to measure the performance of Machine Learning algorithms are: accuracy, true positive rate (TPR), false positive rate (FPR). A classical way of presenting results based on AI is a confusion matrix as show in Table 1.

		Predicted	
		Positive (anomaly)	Negative (non-anomaly)
Actual	Positive	True positive (TP)	False negative (FN)
	Negative	False positive (FP)	True negative (TN)

Table 1: Confusion matrix

Accuracy: The rate of good predictions, whether the labels are positive or negative.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

True Positive Rate (TPR): Indicator of percentage of malicious behaviors detected correctly. A good system has this metric close to 1.

$$TPR = \frac{TP}{TP+FN}$$

False Positive Rate (FPR): Indicator of normal behaviors detected as malicious. A good system has this metric close to 0.

$$FPR = \frac{FP}{TN+FP}$$

However, in reality, they are not really adapted to the problem of malware detection. Indeed, the purpose of AI algorithms in this work is not only to classify between normal/abnormal behavior. In reality, it is impossible to detect all the anomalous behavior coming from the execution of malware because there are periods when the malware occurs as a normal application, and HPC registers cannot register such a difference. Therefore, detecting only a portion of the malware execution (with a high anomaly score) is sufficient to conclude that malware is present. The following metrics are therefore also used to determine the suitability of the algorithms:

1. **Correct malware detection rate:** the number of detected malwares divided by the number of executed malwares.

2. **Detection time:** the time between malware execution and its detection.

3. **Number of false alerts**: after alert aggregation, an alert raised during the execution of benign applications is identified as a false alert.

4.5 Experimentation Results

The proposed pipeline has been implemented and tested on a machine with 2 GTX 1080 Ti graphic cards and an i5-7500 processor with 64 GB of RAM. We train on a data set of 150 000 samples, 100 000 samples are provided to the neural network training and 50 000 are reserved for calibration. The other 5 data sets, each data set has 240 000 samples containing both the normal application data and the malware, are reserved for algorithm evaluation. Each malware is executed 2 times for each data set, corresponding to 10 times of malware execution in total. To eliminate all potential false positives as well as false alerts, the alert aggregation window is set to 10. This means that, if the second anomaly comes from less than 10 samples apart from the first anomaly, these two will be aggregated as one alert. The rules will be applied for the next anomalies as long as they satisfy the condition. If there is not any anomaly coming after the first alert for less than 10 samples, the anomaly will be considered as a false positive and will be ignored.

Algorithm	AE	MLP	LSTM
Accuracy	63.143%	64.249%	66.074%
TPR	9.227%	12.041%	16.592%
FPR	0.179%	0.180%	0.150%
False Alerts Number	0	0	0

Table 2: Comparison of Metrics between the three Algorithms

The results displayed Table 2 show the similarity between the 3 neural networks: all of them are able to detect all malwares. The three algorithms give a low TPR (9.227%, 12.041% and 16.592%) and a good false-positive rate (less than 0.2%). The low TPR can be explained by the fact that only a portion of samples in the malware execution period are classified as anomalies, other samples with anomaly scores lower than the threshold are then predicted as non-anomaly. However, there is always at least one alert is raised during that time, which means all malware are detected with three algorithms. The alert aggregation succeeds to eliminate all false positive anomalies, hence no false alert is raised. Overall, LSTM has the best Accuracy, TPR and FPR.

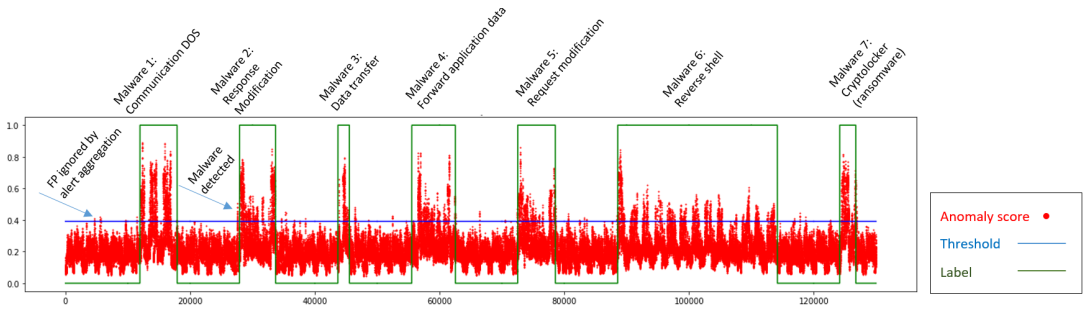

Figure 11: Results on the prediction

Figure 11 pictures the annomaly score computed (red), compared to the labeled data (green) and the detection threshold (blue). The alerts are raised during the whole execution time of malwares 1, 2 and 7, so we can conclude that these malwares are the most impacting to the system. For other malwares, we can see the anomaly score increased at the beginning of execution. This justifies that these malwares are quieter and more difficult to detect because they contain less effective operations. Finally, as shown in Table 3, all malwares are identified under 21 seconds in all cases, and in the best case, they can be detected within 3 seconds. The average detection time varies between 4 and 10 seconds. LSTM has the

Malware	1	2	3	4	5	6	7
True Detection Rate	100%	100%	100%	100%	100%	100%	100%
Best Detection Time (ms)	2560	950	1720	2250	2690	2560	950
Worst Detection Time (ms)	14540	9770	15450	10870	10840	14629	18610
Average Detection Time (ms)	5168	4535	6971	7844	4149	7732	9816

(a) AE (Reconstruction) Detection Results

Malware	1	2	3	4	5	6	7
True Detection Rate	100%	100%	100%	100%	100%	100%	100%
Best Detection Time (ms)	930	2050	1370	1310	1140	1100	1310
Worst Detection Time (ms)	17520	12160	9680	12860	20380	20030	15510
Average Detection Time (ms)	4721	5378	4011	7466	5145	7665	8790

(b) MLP (Prediction) Detection Results

Malware	1	2	3	4	5	6	7
True Detection Rate	100%	100%	100%	100%	100%	100%	100%
Best Detection Time (ms)	2500	1760	2520	1640	2440	2490	700
Worst Detection Time (ms)	8690	10430	9680	10680	11970	11920	14690
Average Detection Time (ms)	3907	3822	5922	7329	3626	6697	5791

(c) LSTM (Prediction) Detection Results

Table 3: Experimentation Results

best average detection time for malware 1, 2, 4, 5, 6, and 7. MLP has the best average detection time for malware 3.

5 Conclusion

In this paper, we introduced the concept of *integrated Monitoring & Recovery Component* (iMRC). This solution allows to address the problem of maintaining a large fleet of IoT devices secure. We described a new architecture in which the iSE can take control over the rest of the SoC and the non-volatile memory. On the cloud side, we showed how a monitoring server can detect malicious behaviors using artificial intelligence. Furthermore, in response to security incidents a first step is to put the device in a degraded mode and regain control of it. Additional data can then be extracted from the application system to analyze the malware. Finally, a patch can be developed and an update deployed to all vulnerable devices of the fleet. Our innovative solution called iMRC improves the resilience of connected objects by enabling the recovery of control of an attacked device. Based on a secure element integrated in the same SoC as the application processor, the iMRC solution itself is intrinsically secure against both software attacks coming from Usys but also against physical attacks targeting Tsys. The use of a remote cloud-based AI for malware detection allows to run powerful AI algorithm without sacrificing on the low resource aspect of embedded devices. Moreover, in case of a loss of connection with the cloud, the devices will automatically switch to a maintenance image to stay safe and can safely be rebooted on the normal image once the connection issue is solve.

We also studied the performance of our malware detection algorithms against a set of tailored malicious scripts reproducing malware behaviors. We introduced our testing methodology on a realistic home automation use case including real software (Home Assistant) alongside off-the-shelf IoT devices. In the future, we plan on experimenting with real malwares, for instance taken from the Juliet data set provided

by the NIST[6]. We would also like to study how moving from an FPGA implementation to an ASIC solution would improve our performance. Finally, we aim to find innovative compression algorithms to optimize the measurement module in order to reduce the size of data sent to the cloud.

References

[1] R. Langner. Stuxnet: Dissecting a cyberwarfare weapon. *IEEE Security & Privacy*, 9(3):49–51, May 2011.

[2] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, A. Halderman L. J, Invernizzi, M. Kallitsis, et al. Understanding the mirai botnet. In *Proc. of the 26th USENIX security symposium (USENIX'17), Vancouver, British Columbia, Canada*, pages 1093–1110. USENIX, August 2017.

[3] S.P. Kadiyala, P. Jadhav, S.K. Lam, and T.Srikanthan. Hardware performance counter-based fine-grained malware detection. *ACM Transactions on Design Automation of Electronic Systems*, 19(5):1–17, September 2020.

[4] A. Marzano, D. Alexander, O. Fonseca, E. Fazzion, C. Hoepers, K. Steding-Jessen, M. Chaves Í. HPC, Cunha, D. Guedes, and W. Meira. The evolution of bashlite and mirai iot botnets. In *Proc. of the 2018 IEEE Symposium on Computers and Communications (ISCC'18), Natal, Brazil*, pages 00813–00818. IEEE, June 2018.

[5] ARM Holdings. Arm security technology: Building a secure system using trustzone technology, April 2009. `https://developer.arm.com/documentation/PRD29-GENC-009492/latest/` [Online; Accessed on May 15, 2022].

[6] D. Andzakovic. Extracting bitlocker keys from a tpm, March 2019. `https://pulsesecurity.co.nz/articles/TPM-sniffing` [Online; Accessed on May 15, 2022].

[7] J. Van Bulck, D. Moghimi, M. Schwarz, M. Lippi, M. Minkin, D. Genkin, Y. Yarom, B. Sunar, D. Gruss, and F.Piessens. Lvi: Hijacking transient execution through microarchitectural load value injection. In *Proc. of the 2020 IEEE Symposium on Security and Privacy (SP'20), San Francisco, California, USA*, pages 54–72. IEEE, May 2020.

[8] S. Guilley, M. Le Rolland, and D. Quenson. Implementing secure applications thanks to an integrated secure element. In *Proc. of the 7th International Conference on Information Systems Security and Privacy (ICISSP'21), online conference*, pages 566–571. SCITEPRESS, February 2021.

[9] Tiempo Secure. Tesic cc eal5+ secure element hard macro. `https://www.tiempo-secure.com/products/tesic-cc-eal-5-secure-element-hard-macro/` [Online; Accessed on May 15, 2022].

[10] Secure-IC. Securyzr ise. `https://www.secure-ic.com/solutions/securyzr/` [Online; Accessed on May 15, 2022].

[11] K. Leung and C. Leckie. Unsupervised anomaly detection in network intrusion detection using clusters. In *Proc. of the 28th ACM Australasian Conference on Computer Science (ACSC'05), Newcastle, Australia*, pages 333–342. ACM, January 2005.

[12] L. Koc, T.A. Mazzuchi, and S. Sarkani. A network intrusion detection system based on a hidden naïve bayes multiclass classifier. *Expert Systems with Applications*, 39(18):13492–13500, December 2012.

[13] A.S. Khan, Z. Ahmad, J. Abdullah, and F. Ahmad. A spectrogram image-based network anomaly detection system using deep convolutional neural network. *IEEE Access*, 9:87079–87093, June 2021.

[14] I. Ullah and Q.H. Mahmoud. Design and development of a deep learning-based model for anomaly detection in iot networks. *IEEE Access*, 9:103906–103926, July 2021.

[15] C.V. Zhou, C. Leckie, and S. Karunasekera. A kangaroo-based intrusion detection system on software-defined networks. *Computer Networks*, 184:107688, January 2021.

[16] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman. Deep learning approach for intelligent intrusion detection system. *IEEE Access*, pages 41525–41550, April 2019.

[17] M. Yousefi-Azar, L.G.C. Hamey, V. Varadharajan, and S. Chen. Malytics: A malware detection scheme. *IEEE Access*, 6:49418–49431, March 2018.

[6] `https://samate.nist.gov/SRD/testsuite.php`

[18] T.G. Kim, B.J. Kang, M. Rho, S. Sezer, and E.G. Im. A multimodal deep learning method for android malware detection using various features. *IEEE Transactions on Information Forensics and Security*, 14(3):773–788, August 2019.

[19] E.B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb. Maldozer: Automatic framework for android malware detection using deep learning. *Digital Investigation*, 24:S48–S59, March 2018.

[20] Z. Ma, H. Ge, Y. Liu, M. Zhao, and J. Ma. Collaborative detection of fast flux phishing domains. *IEEE Access*, 7:21235–21245, January 2019.

[21] F. Mercaldo and A. Santone. Deep learning for image-based mobile malware detection. *Journal of Computer Virology and Hacking Techniques*, 16:1–154, Junuary 2020.

[22] V. Pappas. kbouncer: Efficient and transparent rop mitigation. Technical report, Columbia University, 2012.

[23] I. Fratrić. Ropguard: Runtime prevention of return-oriented programming attacks. Technical report, University of Zagreb, Faculty of Electrical Engineering and Computing, 2012.

[24] Y. Cheng, Z. Zhou, Y. Miao, X. Ding, and R.H. Deng. Ropecker: A generic and practical approach for defending against rop attack. In *Proc. of the 21th Annual Network and Distributed System Security Symposium (NDSS'14), San Diago, California, USA*, pages 23–26. NDSS, February 2014.

[25] Y. Xia, Y. Liu, H. Chen, and B. Zang. Cfimon: Detecting violation of control flow integrity using performance counters. In *Proc. of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'12), Boston, Massachusetts, USA*, pages 1–12. IEEE, August 2012.

[26] Datalog Cloud SIEM. Sqreen. `https://sqreen.com` [Online; Accessed on May 15, 2022].

[27] Baidu. Openrasp. `https://github.com/baidu/openrasp` [Online; Accessed on May 15, 2022].

[28] Red Balloon Security. Runtime protection. `https://redballoonsecurity.com/` [Online; Accessed on May 15, 2022].

[29] HDIV. Hdiv protection. `https://hdivsecurity.com/` [Online; Accessed on May 15, 2022].

[30] Imprevas. Application security. `https://www.imperva.com/` [Online; Accessed on May 15, 2022].

[31] Application security. Karamba security. `https://karambasecurity.com/` [Online; Accessed on May 15, 2022].

[32] Checkpoint Security. Zero-day protection: Block unknown threats. `https://www.checkpoint.com/fr/` [Online; Accessed on May 15, 2022].

[33] SternumIoT. Embedded integrity verification. `https://www.sternumiot.com/` [Online; Accessed on May 15, 2022].

[34] 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 *Proc. og the 2019 IEEE Symposium on Security and Privacy (SP'19), San Francisco, California, USA*, pages 20–38. IEEE, May 2019.

[35] J. Demme, M. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan, and S. Stolfo. On the feasibility of online malware detection with performance counters. *ACM SIGARCH Computer Architecture News*, 41(3):559–570, June 2013.

[36] X. Wang and R. Karri. Numchecker: Detecting kernel control-flow modifying rootkits by using hardware performance counters. In *Proc. of the 50th ACM/EDAC/IEEE Design Automation Conference (DAC'13), Austin, Texas, USA*, pages 1–7. IEEE, Jun 2013.

[37] A. Tang, S. Sethumadhavan, and S.J. Stolfo. Unsupervised anomaly-based malware detection using hardware features. In *Proc. of the 17th International Symposium International Workshop on Recent Advances in Intrusion Detection (RAID'14), Gothenburg, Sweden*, volume 8688 of *Lecture Notes in Computer Science*, pages 109–129. Springer-Verlag, September 2014.

[38] A. Garcia-Serrano. Anomaly detection for malware identification using hardware performance counters. Technical report, Universitat Oberta de Catalunya, 2015.

[39] H.W. Zhou, X. Wu, W.C. Shi, J.H. Yuan, and B. Liang. Hdrop: Detecting rop attacks using performance monitoring counters. In *Proc. of the 10th International Conference on Information Security Practice and Experience (ISPEC'14), Fuzhou, China*, volume 8688 of *Lecture Notes in Computer Science*, pages 172–186. Springer-Verlag, May 2014.

[40] M. Bahador, M. Bagher, Abadi, and A. Tajoddin. Hpcmalhunter: Behavioral malware detection using hard-

ware performance counters and singular value decomposition. In *Proc. of the 4th IEEE International Conference on Computer and Knowledge Engineering (ICCKE'14), Mashhad, Iran*, pages 703–708. IEEE, October 2014.

[41] X. Wang and J. Backer. Sigdrop: Signature-based rop detection using hardware performance counters. Technical report, Tandon School of Engineering, New York University, 2016.

[42] S. Das, B. Chen, M. Chandramohan, Y. Liu, and W. Zhang. Ropsentry: Runtime defense against rop attacks using hardware performance counters. *Computers & Security*, 73(1):374–388, March 2018.

[43] X. Wang, S. Chai, M. Isnardi, S. Lim, and R. Karri. Hardware performance counter-based malware identification and detection with adaptive compressive sensing. *ACM Transactions on Architecture and Code Optimization*, 13(1):1–23, April 2016.

[44] H. Peng, J. Wei, and W. Guo. Micro-architectural features for malware detection. In *Proc. of the 11th Conference on Advanced Computer Architecture (ACA'16), Weihai, China*, pages 48–60. Springer-Verlag, August 2016.

[45] V. Jyothi, X. Wang, S.K. Addepalli, and R. Karri. Brain: Behavior based adaptive intrusion detection in networks: Using hardware performance counters to detect ddos attacks. In *Proc. of the 29th IEEE International Conference on VLSI Design and IEEE 15th International Conference on Embedded Systems (VLSID'16), Kolkata, India*, pages 587–588. IEEE, March 2016.

[46] M. Chiappetta, E. Savas, and C. Yilmaz. Real time detection of cache-based side-channel attacks using hardware performance counters. *Applied Soft Computing*, 49:1162–1174, 2016.

[47] T. Zhang, Y. Zhang, and R.B. Lee. Dos attacks on your memory in cloud. In *Proc. of the 2017 ACM on Asia Conference on Computer and Communications Security (Asia CCS'17), Abu Dhabi, United Arab Emirates*, pages 253–265. ACM, April 2017.

[48] B. Singh, D. Evtyushkin, J. Elwell, R. Riley, and I. Cervesato. On the detection of kernel-level rootkits using hardware performance counters. In *Proc. of the 2017 ACM on Asia Conference on Computer and Communications Security (Asia CCS'17), Abu Dhabi, United Arab Emirates*, pages 483–493. ACM, April 2017.

[49] N.F. Polychronou, P.H. Thevenon, V.M. Puys, and Beroulle. A comprehensive survey of attacks without physical access targeting hardware vulnerabilities in iot/iiot devices, and their detection mechanisms. *ACM Transactions on Design Automation of Electronic Systems*, 27(1):1–35, September 2021.

[50] C. Li and J.L. Gaudiot. Detecting malicious attacks exploiting hardware vulnerabilities using performance counters. In *Proc. of the 43th IEEE Annual Computer Software and Applications Conference (COMPSAC'19), Milwaukee, Wisconsin, USA*, pages 588–597. IEEE, September 2019.

[51] B. Gülmezoglu, A. Moghimi, T. Eisenbarth, and B. Sunar. Fortuneteller: Predicting microarchitectural attacks via unsupervised deep learning. *Clinical Orthopaedics and Related Research*, abs/1907.03651, July 2019.

[52] N.F. Polychronou, P.H. Thevenon, M. Puys, and V. Beroulle. Madman: Detection of software attacks targeting hardware vulnerabilities. In *Proc. of the 24th IEEE Euromicro Conference on Digital System Design (DSD'21), Palermo, Italy*, pages 355–362. IEEE, September 2021.

[53] S. Das, Y. Liu, W. Zhang, and M. Chandramohan. Semantics-based online malware detection: Towards efficient real-time protection against malware. *IEEE Transactions on Information Forensics and Security*, 11(2):289–302, February 2015.

[54] MCUboot. Secure boot for 32-bit microcontrollers. `https://www.mcuboot.com/` [Online; Accessed on May 15, 2022].

[55] J.S. Plank, M. Beck, G. Kingsley, and K. Li. Libckpt: Transparent checkpointing under unix. In *Proc. of the 1995 USENIX Technical Conference Proceedings (TCON'95), New Orleans, Louisiana, USA*, page 18. USENIX Association, January 1995.

[56] B. Ransford, J. Sorber, and K. Fu. Mementos: System support for long-running computation on rfid-scale devices. In *Proc. of the 16th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'11), Newport Beach, California, USA*, pages 159–170. ACM, May 2011.

[57] T. Li, R. Ragel, and S. Parameswaran. Reli: Hardware/software checkpoint and recovery scheme for embed-

ded processors. In *Proc. of the 2012 IEEE Conference on Design, Automation & Test in Europe Conference & Exhibition (DATE'12), Dresden, Germany*, pages 875–880. IEEE, March 2012.

[58] A. Mirhoseini, E.M. Songhori, and F. Koushanfar. Idetic: A high-level synthesis approach for enabling long computations on transiently-powered asics. In *Proc. of the 2013 IEEE International Conference on Pervasive Computing and Communications (PerCom'13), San Diego, California, USA*, pages 216–224. IEEE, March 2013.

[59] H. Jayakumar, A. Raha, and V. Raghunathan. Quickrecall: A low overhead hw/sw approach for enabling computations across power cycles in transiently powered computers. In *Proc. of the 27th IEEE International Conference on VLSI Design (VLSID'14) and 13th International Conference on Embedded Systems (EMSOFT'14), Mumbai, India*, pages 330–335. IEEE, January 2014.

[60] C. Wang, L. Gong, Q. Yu, X. Li, Y. Xie, and X. Zhou. Dlau: A scalable deep learning accelerator unit on fpga. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, (3):513–517, JulY 2016.

[61] X. Wang, Z. Zhao, D. Xu, Z. Zhang, Q. Hao, M. Liu, and Y. Si. Two-stage checkpoint based security monitoring and fault recovery architecture for embedded processor. *Electronics*, 9(7):1165, July 2020.

[62] NIST. Cybersecurity framework's five functions, April 2018. `https://www.nist.gov/cyberframework/online-learning/five-functions` [Online; Accessed on May 15, 2022].

[63] O. Aslan and R. Samet. A comprehensive review on malware detection approaches. *IEEE Access*, 8:1–1, January 2020.

[64] M. Labonne, A. Olivereau, B. Polvé, and D. Zeghlache. Unsupervised protocol-based intrusion detection for real-world networks. In *Proc. of the 2020 International Conference on Computing, Networking and Communications (ICNC'20), Big Island, Hawaii, USA*, pages 299–303. IEEE, 2020.

[65] infracritical Finding reports. Project shine (shodan intelligence extraction), 2014. `https://www.slideshare.net/BobRadvanovsky/project-shine-findings-report-dated-1oct2014` [Online; Accessed on May 15, 2022].

[66] infracritical Finding reports. Project ruggedtrax, 2015. `https://www.slideshare.net/BobRadvanovsky/ruggedtrax-findings21oct2014prelim` [Online; Accessed on May 15, 2022].

[67] N.F. Polychronou, P.H. Thevenon, M. Puys, and V. Beroulle. Securing iot/iiot from software attacks targeting hardware vulnerabilities. In *Proc. of the 19th International New Circuits and Systems Conference, (NEWCAS'21), Toulon, France*, pages 1–4. IEEE, June 2021.

[68] A.F. Agarap. Deep learning using rectified linear units (relu). Technical report, Computer Science, University of Cornell, 2018.

[69] C. Sammut and G.I. Webb. Mean squared error. In *Encyclopedia of Machine Learning*, page 653. Springer Verlag, 2010.

Author Biography

Pierre-Henri Thevenon is a research engineer at the LETI institute, CEA (Commissariat à l'énergie atomique et aux énergies alternatives), specialized in the security of IoT and I-IoT. After receiving his Ph.D in 2011 from University of Grenoble Alpes on the security of contactless systems, he worked on the development and security of embedded systems. For the last 4 years, he has been leading multiple Industrial IoT Security projects, within IRT Nanoelec.

Sebastien Riou works as Application Manager at Tiempo Secure. Previously he worked over a decade in the smart card industry in various technical roles. He submitted several patents related to secure IC implementation. As a freelance penetration tester, he helped finding vulnerabilities in consumer products and a whitebox cryptography product. He designed the algorithm DryGASCON, a round2 candidate in the "LWC" competition run by NIST. Nowadays Sebastien helps customers to define what security means in the context of their product and how to implement it.

Duc-Minh Tran is a research engineer at the LIST Institute, CEA (Commissariat à l'énergie atomique et aux énergies alternatives). He received his MSc in Cybersecurity from INSA Centre Val de Loire in 2020. He is now in charge of the development of a network intrusion detection system called SIGMO-IDS. His main areas of research are oriented around network security, anomaly detection system using Machine Learning, autonomous reaction against cyberattacks.

Maxime Puys is a research engineer at the LETI Institute, CEA (Commissariat à l'énergie atomique et aux énergies alternatives). He received his Ph.D in 2018 from University of Grenoble Alpes. His thesis focused on the cybersecurity of industrial control systems against network attacks and risk analysis combining safety and security. He now designs security components and contributes to the development of new tools to analyze the security of Industrial IoT.

Nikolaos-Foivos Polychronou is a PhD student at the LETI Institute, CEA (Commissariat à l'énergie atomique et aux énergies alternatives) and the University of Grenoble Alpes since December 2019. He received his MSc in Engineering from National Technical University of Athens in 2018. In his PhD thesis, he aims at building a solution to detect malware in IoT and IIoT devices, focusing on the detection of software attacks targeting hardware vulnerabilities. He is also experienced with side channel attacks.

Mustapha El Majihi is a research engineer at the LETI Institute, CEA (Commissariat à l'énergie atomique et aux énergies alternatives). He graduated as an engineer in electronics from ENSEEIHT Toulouse INP in 2012 and received his MSc in signal and image processing from Grenoble INP in 2014. He mainly works on securing microarchitectures based on the RISCV instruction set by developing and implementing countermeasures against hardware and software vulnerabilities.

Camille Sivelle is a research engineer at the LETI Institute, CEA (Commissariat à l'Énergie Atomique et aux Énergies Alternatives), specialized in IoT cybersecurity. She received her MSc in computer science from Lorraine University in 2020 where she was dealing with formal analysis of cryptographic protocols. She joined CEA in 2020 where she now carries out research on firmware emulation, malware detection and cryptographic protocols code generation.