IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 20 October 2023, accepted 14 November 2023, date of publication 20 November 2023,
date of current version 29 November 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3335187

SURVEY

Toward Hardware-Assisted Malware Detection Utilizing Explainable Machine Learning: A Survey

YEHYA NASSER[1] **AND MOHAMAD NASSAR**[2]**, (Member, IEEE)**
[1]Lab-STICC, UMR CNRS 6285, IMT Atlantique School, 29238 Brest, France
[2]Department of Computer Science, The University of Alabama in Huntsville, Huntsville, AL 35899, USA

Corresponding author: Yehya Nasser (yehya.nasser@imt-atlantique.fr)

This work was supported in part by Direction Générale de L'armement Française and Région Bretagne.

ABSTRACT Hardware joined the battle against malware by introducing secure boot architectures, malware-aware processors, and trusted platform modules. Hardware performance indicators, power profiles, and side channel information can be leveraged at run-time via machine learning for continuous monitoring and protection. The explainability of these machine learning algorithms may play a crucial role in interpreting their results and avoiding false positives. In this paper, we present an eagle eye on the state of the art of these components: we examine secure architectures and malware-aware processors, such as those implemented in the RISC-V Instruction Set Architecture and Reduced Instruction Set Computer (RISC). We categorize hardware-assisted solutions increased by machine learning for classification. We survey recently proposed software-assisted and hardware-assisted explainability algorithms in our context. In the discussion, we suggest that (1) safe architectures that guarantee secure device boot are a must, (2) Side-channel approaches are challenging to integrate into embedded systems, yet they show promise in terms of efficiency, (3) malware-aware processors provide valuable features for malware detection software, and (4) Without explainability, malware detection software is error-prone and can be easily bypassed.

INDEX TERMS Hardware security, embedded systems, malware detection, secure boot, explainability, machine learning, side channels, IoT.

I. INTRODUCTION

According to a study by Cisco [1], 500 billion electronic devices are expected to be connected to the Internet by 2030. The proliferation of the Internet of Things (IoT) has brought about a significant surge in the deployment of small, connected embedded devices. However, these devices are particularly vulnerable to attacks due to their inherent limitations in computational power, resources, and power consumption constraints. Consequently, one of the primary concerns surrounding embedded systems within the IoT ecosystem is the potential disruption of their functionality by a malware program [2]. Malicious software or Malware targets these systems with the intention of modifying their expected behavior, which can lead to various detrimental outcomes such as unauthorized data exfiltration, impaired device functionality, or transforming the device into a controlled bot for the attacker's purposes. Malware includes harmful software such as viruses, ransomware, miners, etc. Antivirus software counters malware via signature or behavior detection. However, it can miss new or unknown variants, hence the need for regular updates and cautious online habits. Detection failures can be due to mutated or evolving malware [3].

Hardware-assisted malware detection, which identifies malicious software by monitoring the hardware footprints of its operations, is a method that utilizes dedicated hardware to obtain an execution profile, thereby likely enhancing malware detection [4]. It is also possible to utilize these profiles with machine learning-based algorithms to recognize patterns in the execution of the program that may bring information about a possible malware execution, allowing them to detect even previously unseen types of malware. Hardware-assisted malware detection can be implemented within various hardware platforms, including General-Purpose Processors (GPPs), Graphics Processing Units (GPUs), and specialized

The associate editor coordinating the review of this manuscript and approving it for publication was Jiafeng Xie.

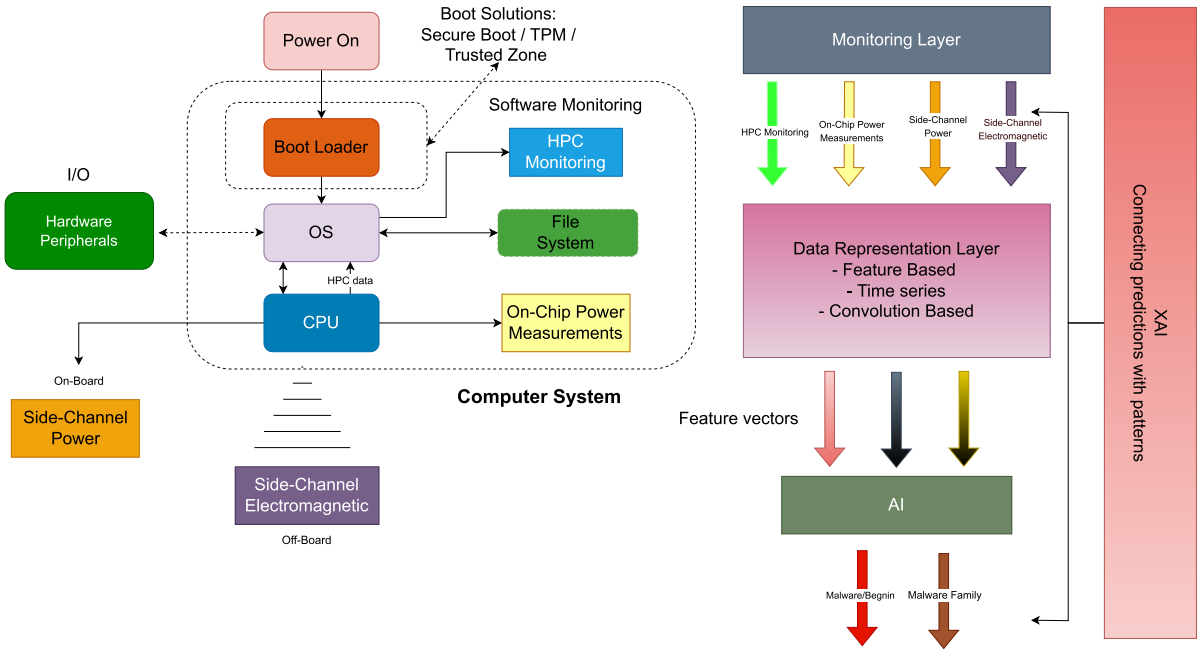

FIGURE 1. Points of interest for malware detection and prevention in embedded computer systems.

hardware accelerators. Hardware-assisted malware detection can enhance performance, diminish false positives, and bolster security in comparison to software approaches [5]. Moreover, it is expected to be immune to forms of software manipulation [6].

To this end, cost-effective, energy- and area-saving solutions for hardware-assisted malware detection are needed. Although this article surveys and discusses several solutions that use machine learning and hardware capabilities to detect and mitigate malware execution, we also cover some solutions that use side-channel information such as power consumption. Finally, we discuss the explainability of machine learning models to understand whether these models are genuinely effective in detecting malware. The scope of the discussion is depicted through the points of interest as shown in Figure 1. The contributions of this paper are summarized as follows:

- Presenting an overview of hardware-assisted malware detection using machine learning and explainability.
- Discussing the potential issues and trends in the field of hardware-assisted malware detection and its future developments.

The rest of the paper is organized as follows: Section II serves as a background on the various topics explored in this paper. In Section III, we present an in-depth overview of hardware architecture solutions employed for the purpose of malware detection. Section IV focuses on explainable machine-learning models for malware detection. Moving on to Section V, we discuss state-of-the-art solutions, their categorization, and merits. Finally, Section VI addresses the challenges and future directions for malware detection, besides explainability. Section VII concludes the paper.

II. BACKGROUND

A. MALWARE DETECTION AND MITIGATION

The aim of malware can be to retrieve private information, gain remote access to the machine, or encrypt the user's memory for profit. Malware can be hidden in documents that victims may receive in email attachments or downloaded software, and can spread through the exploitation of vulnerabilities in firmware or applications for embedded devices. While it is beyond the scope of this paper to enumerate all types of malware (e.g., botnet, adware) and their delivery modes, it is essential to note that malware can have devastating consequences for companies, especially when dealing with ransomware. According to [7], there are more than 1.1 billion malware programs in existence. At the same time, as IoT and industrial IoT applications continue to increase, the number of malware instances has increased by 77% in 2022 compared to 2021. The traditional approach to mitigating malware is to use antivirus software. This software continuously scans the computer in search of any malware and quarantines or removes it. Traditional malware detection techniques can be static, which analyzes the structure of a program, or dynamic, which analyzes the behavior at run-time. Malware detection can focus on signatures of known malware or detecting deviations from the regular behavior of the system [8]. Other techniques that can be classified as best practices can also be used to avoid malware execution. For example, it is highly recommended to always patch and update software and systems for vulnerability fixes. Encrypting and isolating memory can also be a way to prevent malware from disrupting essential data on the system [9]. In addition, securing networks is critical; for example, the network administrator may install

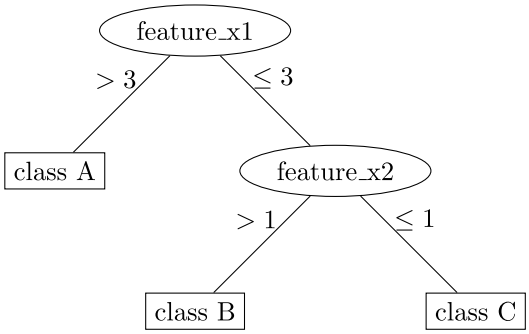

FIGURE 2. Decision tree structure.

firewalls, hardware or software, to prevent malware from entering the network. They can also segment networks into small and isolated subnets. Last but not least, virtualization, thanks to the hardware-assisted virtualization features of modern processors, allows the user to create isolated virtual environments that will not compromise the host machine if infected by malware [10]. Some of these solutions are costly regarding computer performance and, therefore, unsuitable for constrained devices such as embedded systems or IoT devices. Thus, our analysis takes into account the target platform and applications.

B. MACHINE LEARNING

Machine learning is the field of study that allows computers to learn a task without being explicitly programmed. Machine learning can be divided into three categories:

- Supervised Learning: Learning is made using data consisting of input values and the corresponding outputs. Such tasks include regression or classification.
- Unsupervised Learning: Learning uses data that consists only of input values without the corresponding outputs. Such tasks include clustering, density estimation, dimensionality reduction, and visualization.
- Reinforcement learning: Learning a policy composed of pairs (state, action) based on trails of actions and rewards.

For example, a decision tree is a sequential decision model that uses a tree-like structure, as presented in Figure 2.

Decision trees are a commonly used machine learning method because they are explainable by design. Each tree node divides the data according to a criterion learned during training. The user can retrieve the sequence of met conditions to reach a prediction simply by examining the path from the root node to the leaf node of the decision.

Neural networks and deep learning are methods inspired by biological brains that have proven outstanding performance for various tasks such as computer vision, voice analysis, natural language processing, etc. Neural networks comprise layers of neurons that can be fully connected or share parameters. A single neuron implements a biased weighted sum of its inputs followed by a non-linear transform:

$$y = f\left(\sum_{k=1}^{N} w_k x_k + b\right)$$

where:

- y: is the output of the neuron,
- w_k: is an input weight,
- x_k: is an input feature,
- b: is the bias,
- f: is a non-linear function such as ReLu or Sigmoid.

Neurons can be connected to form complex neural networks with up to billions of parameters in total. The most recent neural networks are reported to have a trillion parameters.

C. EXPLAINABILITY

New algorithms consider the explainability of deep neural networks. Explainability can be defined by the information used to help the user understand the decision of the neural network [11]. Formally, an explanation can take several definitions. For tabular data, the explanation is a set of golden (feature, value) pairs. For a picture, the explanation is one or more regions or a set of pixels. For a text, the explanation is a set of words, groups of words, or sentences. For a signal, it can be a time frame or a set of clock cycles, as suggested in [12]. In all cases, an explanation is an attempt to highlight the part of the input data that causes the classifier to return a specific decision. Other forms of explanation include anchors [13] and counterfactuals [14].

There are several ways to obtain forms of explainability from a neural network. We can divide them into three main categories: black-box methods, white-box methods, and explainability by design.

In black box methods, no properties about the architecture, the training data, or the neural network weights are used to extract explainability. Black-box methods can involve the use of a surrogate explainable model that locally approximates the function of the "unexplainable" deep neural network. We can refer to the LIME method (Local Interpretable Model-agnostic Explanations) [15]. The design of the surrogate model introduces a fidelity-interpretability trade-off because a more comprehensible explanation might be less representative of the deep network. Usually, these approximate models are designed based on linear models or decision trees and, more generally, any interpretable design model [16].

More formally:

$$\epsilon(\mathbf{x}) = \arg\min_{g \in G} \{L(f, g, \pi_{\mathbf{x}}) + \Omega(g)\}$$

where

- $\mathbf{x}$ is the data point at which the model is interpreted,
- $\epsilon(\mathbf{x})$ is the surrogate model,
- $g \in G$ is a model which belongs to a class of interpretable models G,
- $\Omega(g)$ is a measure of the capacity of g.

- f, the overall deep learning model of which we want to explain one prediction,
- $\pi_{\mathbf{x}}$ is a proximity measure of x that defines locality around $\mathbf{x}$,
- L is the loss function, measuring how f and g compares around $\pi_{\mathbf{x}}$.

For example, the class of interpretable models can be sparse linear models. Perturbations scan the neighborhood around the data point of interest.

LEMNA (Local Explanation Method using Non-Linear Approximations) [17] is a method built as a high-fidelity explanation method for security applications. It achieves high fidelity by using non-linear approximations of the surrogate model (contrary to the LIME method that uses linear approximations). Using a fused Lasso loss function also enforces feature dependency for time series features. A comparison of LIME and LEMNA for two applications: (1) the identification of function starts in binaries, and (2) the classification of PDF malware represented as feature vectors shows that LEMNA outperforms LIME in both applications.

Contrary to black-box methods, all neural network properties can be used in white-box methods, such as intermediate-layer activations or gradient values.

The goal of explainability by design is to modify the properties of the neural network prior to its training, such as its architecture or cost function, to make the model self-explainable. For example, some machine learning models, such as decision trees, are intrinsically explainable.

III. HARDWARE-ASSISTED MALWARE DETECTION

We present four approaches for hardware-assisted malware detection and prevention: secure boot architectures, on-board power measurements, on-chip performance indicators, and off-board side channels.

A. SECURE BOOT: SECURE ARCHITECTURES

Secure boot is a crucial defense mechanism that safeguards systems against malicious code execution throughout the boot sequence. Its primary function is to protect the integrity of a digital system, which is essential to fend off malicious attacks. The mechanism operates by validating the authenticity of each boot sequence stage, which encompasses the BIOS, bootloader, and Operating System (OS). It accomplishes this by calculating the hash value of the component code and juxtaposing it with a trusted signature (unmodified code). Any discrepancies trigger a halt in the system's execution, thereby fortifying it against prospective malware threats. For example, malware could infiltrate the BIOS (Basic Input/Output System) or UEFI (Unified Extensible Firmware Interface), the firmware that initializes the system before running any software. This level of infection poses a grave threat, as malware can survive system reboots and even persist following OS reinstall, thereby attaining low-level system access to potentially compromise the system further [18].

Secure boot is a security protocol developed by computer industry professionals, aiming to ensure a device is launched using only software approved by the Original Equipment Manufacturer (OEM). Upon starting, the device's firmware verifies the signature of each component involved in the booting process, such as UEFI firmware drivers, EFI applications, and the operating system. If these signatures are authenticated, the device boots and gives control to the operating system. The OEM can generate secure boot keys following instructions provided by the firmware manufacturer and store them in the device's firmware. When introducing UEFI drivers, it is crucial to ensure that they are signed and added to the Secure Boot database [19]. When a system is using secure boot, each piece of boot software is accompanied by a digital signature. This includes the bootloader and the operating system. These signatures are then compared to the keys stored in the firmware. If the keys match, the system boots up. If not, the system will stop booting. There are various types of secure boot solutions:

- Recoverable Malware Protection: In this method, the secure boot process includes the ability to recover from a detected malware attack [20]. This could mean a reset to factory settings or a rollback to a previous safe state, depending on the particular implementation.
- Multilayered Security Features: This refers to the use of several security measures together to protect the boot process [21]. This could include bootloaders that verify each other, multiple checks of firmware signatures, hardware-based security measures, and more.
- ARM Trust Zone technology is a system-wide approach to security for a wide array of client and server computing platforms, including handhelds, mobile devices, tablets, and servers. In ARM architecture, the secure boot process is used to validate the authenticity of the code before it is loaded into the environment [22]. During secure boot in ARM Trust Zone, the processor starts in a safe state, and once the secure boot phase is finished, the non-secure world is allowed to execute. This ensures that a known environment has been established before the less trusted portions of the system are allowed to boot.

However, it is essential to note that a secure boot is not a complete security solution in itself. It is a part of a larger security strategy, and it is only effective at preventing unauthorized code from running at boot time. Other security measures are necessary to protect a system during its operation.

RISC-V (pronounced "risk-five") is an open standard Instruction Set Architecture (ISA) based on established RISC principles. Unlike proprietary ISAs such as Intel's x86 or ARM, RISC-V is open source, which means it's freely available to the public and can be used freely in any computing device.

The RISC-V ISA has been designed with several key advantages in mind. These include a smaller and simpler set of instructions, leading to more efficient CPU design and

FIGURE 3. Placement of CARE [20].

lower power consumption. It also has appealing features for security applications, such as support for a wide range of security models and cryptographic algorithms.

Given its open-source nature, RISC-V provides a great deal of flexibility and allows custom optimizations not possible with proprietary ISAs. It enables developers to design their own extensions or custom instructions, making it possible to create highly specialized and optimized hardware implementations.

CARE and ITUS are the secure boot hardware-firmware solutions implemented in RISC-V that are presented in the provided context because they both have implemented secure boot strategies, a critical feature for ensuring system security. By discussing their implementation in the RISC-V architecture, the passage emphasizes the suitability of RISC-V for secure applications. The specific implementations of secure boot in CARE and ITUS could highlight the benefits and potential of RISC-V in creating secure systems. To this end, in this next section, we introduce two solutions designed using RISC-V, namely, CARE and ITUS.

1) SECURE BOOT WITH RECOVERY

The Code Authentication and Resilience Engine (CARE) is a hardware-based secure boot solution designed for the RISC-V system [20]. Its hardware implementation improves security compared to software-based solutions. CARE protects against rootkit malware (malicious programs that gain unauthorized control over a computer while concealing their activities) and BIOS and secure boot attacks (attacks that exploit firmware to run unauthorized code during a system's boot process).

It is good to mention here that conventional secure boot protocols only halt the boot sequence upon detection of an integrity anomaly, requiring software re-flashing, but here CARE provides recovery of the system. As shown in Figure 3 CARE is placed between the first stage boot code of the ROM and the second stage boot code of the flash. CARE relies on a secure SPI (Serial Peripheral Interface) bus to communicate with ROM and Flash, and it securely stores data, like signing keys and device information, in the ROM (with specific access policies), which is critical for the device's recovery and security.

As depicted in Figure 4, CARE has two main components: the code integrity and authentication (CA) unit and the resilience engine (RE). When the system is powered up, the first-stage bootloader initializes the SPI and flash controllers and applies the memory protection rules. The control is then passed to the second stage boot code (bootstrap). The flash code is divided into frames and sent to the CA over the SPI

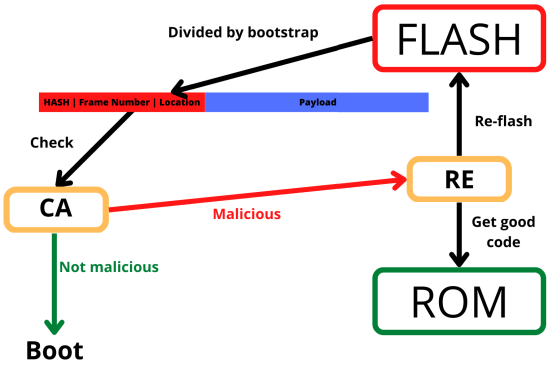

FIGURE 4. Secure boot with CARE [20].

bus. The frame is composed of a header with the signed digest of the data frame, the frame number, and the flash memory location; the rest of the frame is the payload. The CA has a lightweight cryptographic core (HMAC-SHA256), the SHA256 module computes the digest of each frame, and the HMAC-SHA256 uses a derived key to sign it. The calculated digest is finally compared to the hash in the header. If malicious code is detected, the resilience engine is activated.

The Resilience Engine (RE) identifies the frame number and flash location of the corrupted frame and then locates the backup in the EEPROM. Subsequently, the RE reflashes the flash memory with the known good code and locks unauthorized read-write access to the memory. The CARE method assumes that the code stored in ROM, typically written during manufacturing, cannot be easily modified, assuring against common malware attacks that might alter the boot code.

Finally, CARE is designed to address the security risks that arise when malware compromises the flash, as it could also interfere with the ROM reload process, potentially introducing malicious code during reload. While, theoretically, the flash can be automatically reloaded from ROM as the initial step of the secure boot, this approach can introduce significant operational overhead, especially for systems with large flash memories or where boot speed is critical. Moreover, CARE provides additional flexibility and recovery options that simple reloading may not offer. For example, CARE can identify and recover from issues at a granular level (i.e., frame level) without requiring a complete reload of the entire flash memory, which can be beneficial in scenarios where quick recovery is essential or where network conditions might make fetching and reloading the whole code-base impractical. Furthermore, CARE can mitigate more advanced threats by providing an additional layer of security through its Code Integrity and Authentication (CA) unit and the Resilience Engine (RE). This unit allows CARE to detect and recover from problems in a controlled and secure manner, which is impossible with a simple automatic reload mechanism.

2) BOOT WITH BOOSTED SECURITY FEATURES

Another solution worth studying is ITUS[1] [21]. ITUS is implemented in LowRISC. LowRISC is an open-source SoC platform built around the open-source RISC-V ISA. ITUS aims to provide a trusted system through several security mechanisms, including secure boot, encryption, and authentication of off-chip memory accesses, key management, and cryptography acceleration. Those security features make it possible to create enclaves, which are isolated trusted execution environments used to protect sensitive data and processes.

ITUS focuses on mitigating attacks that aim to gain unauthorized access to sensitive data, such as Rowhammer, Meltdown, or Foreshadow. ITUS is composed of different blocks:

- *Asymmetric cryptography*: enables RSA and elliptic curve algorithms for secure boot, remote attestation, remote software updates, and digital signatures.
- *Symmetric cryptography*: ensures data encryption and integrity during storage and data transmission, and prevents reverse engineering.
- *Physical Unclonable Function (PUF)*: allows a systematic approach to creating a unique device ID during manufacturing.
- *True Random Number Generator (TRNG)*: is used to generate ephemeral session keys and seed values for encryption.
- *Root-of-trust (ROT)*: a Trusted Execution Environment (TEE) [23] is a hardware clearly defined security perimeter. We can find three types of TEE: one that allows running some application modules in an encrypted enclave, one that runs virtual machines in encrypted memory, and one that is hardware isolation for secure computing.
- *Chain-of-Trust*: divides the system into different levels that are checked starting from the hardware, then the bootloader, BIOS/firmware, the OS, and finally the application layer, to ensure that all these parts can be trusted.

It is good to mention here that security features can be implemented with these different blocks to provide a secure and reliable system. For instance, there is the secure boot, which checks the integrity of each layer of the boot process by computing hashes. This process has already been described with CARE, but we do not have a recovery option here.

Another important security feature is memory protection, as TEE must communicate with untrusted external storage. This feature requires a memory protection unit (MPU) to manage reading and writing to the trusted region. The MPU is responsible for preserving the confidentiality and integrity of the transferred data. With ITUS, this is implemented using the AES CGM encryption scheme. Other methods for securing memory include randomization of the address space layout against overflow and control flow hijacking [24], automatic constant time programming against micro-architectural side channels [25], and oblivious RAM [26].

Last but not least, ITUS is capable of managing keys. Permanent and device-specific keys can be generated thanks to PUF, and ephemeral keys can use the TRNG. Public keys are signed by a certification authority for permanent device-specific keys, and device keys can sign ephemeral keys if needed.

In summary, CARE and ITUS implement a secure on-chip architecture without machine learning. They efficiently detect malware and ensure the device is in a trusted state before using it. However, we have yet to address the detection of malware at run-time. Also, the performance overhead due to cryptographic operations in microarchitectural security management is non-negligible, and achieving security with minimal performance cost remains an important area for future work.

3) SECURE BOOT AGAINST ICS MALWARE

Secure boot, a foundational security mechanism, offers a robust defense against unauthorized execution of malicious software in Industrial Control Systems (ICS). This ensures that only software with a recognized cryptographic signature is executed, enhancing the system's resistance against malware intrusions [27], [28].

One of the most sophisticated malware, Stuxnet, discovered in 2010, primarily targeted industrial control systems, specifically those associated with Siemens' Supervisory Control and Data Acquisition (SCADA) systems [29], [30]. Its propagation methods included the compromise of USB drives, leading to the execution of malware after its insertion into a system. By ensuring the integrity of the boot process, secure boot thwarts such malware from executing its malicious payload, especially if it attempts to modify boot or system files.

Another notable incident, the Jeep Hack of 2015, saw security researchers remotely manipulating a Jeep Cherokee's functionalities by exploiting vulnerabilities in its entertainment system [31]. While secure boot primarily safeguards the boot process, the overarching principle of executing only trusted software can be extended to other systems. In vehicular contexts, if the critical systems were isolated and restricted to run only signed, trusted software, it would significantly reduce the risk of such remote attacks.

However, while a secure boot offers a formidable line of defense, it has challenges. Implementing secure boot, whether in hardware or software, can introduce performance overheads. This translates to increased area requirements in hardware, while in software, it can lead to increased power consumption [32].

As we have seen before, CARE and ITUS present two solutions built on top of the RISC-V ISA. These techniques can be compared using a Trusted Platform Module (TPM) [33]. A TPM is a hardware security device that establishes a

[1] ITUS is a name from Greek mythology, not an abbreviation.

root of trust for the system. It can handle attestation, perform cryptographic operations, and thus be used to ensure a secure boot.

The ARM architecture is widely recognized for its energy efficiency and is among the most popular for IoT applications. It incorporates a TPM variant known as the ARM Trust Zone. This TPM aims to segregate the system's resources, both hardware and software, into two distinct domains: the Secure World (SW) and the Normal World (NW). The ARM Trust Zone can be used to ensure the integrity of the system at both boot and run-time [34].

A secure boot is conducted on the SW via a chain of trust mechanism during the boot process. Subsequently, the SW ensures a trusted boot for the NW, implementing measurements of the boot images and remote attestation. As for run-time, the SW performs periodic measurements of processes and sends them to attestation servers for integrity verification against reference values.

Regarding malware prevention, a TPM can also be utilized to mitigate malware threats. Depending on the application, the TPM's cryptographic capabilities and critical information storage can ensure that the operations performed on a system originate from the intended users [35].

However, employing a TPM can be costly in terms of power consumption and cost, making it sometimes unsuitable for all IoT devices. An alternative solution is measured boot, a critical security feature that verifies the chain of trust of a device by measuring each stage of its firmware and software. This can be deployed, for instance, for the Secure World (SW) boot within the ARM Trust Zone. Although the measured boot process can be performed using a TPM, an efficient alternative is the Device Identifier Composition Engine (DICE) [36]. The DICE engine is the first component to be started. It uses a unique device secret, a hardware secret, to ensure the integrity of the first boot sequence stage. Subsequently, each stage generates keys to verify the next stage of the boot sequence.

Each of these solutions carries its benefits and drawbacks. TPMs are already widely adopted in the industry; however, they require developing unique security applications, including secure boot. Additionally, TPMs also bring about additional power consumption and cost overheads. Owing to these challenges, the Device Identifier Composition Engine (DICE) is often used as a standard to ensure a measured boot of the system, reducing the impact on energy consumption. However, the DICE engine cannot be universally implemented, as it requires the manufacturer to generate protected secrets, such as a unique device secret and specific certificates.

On the other hand, ITUS and CARE present promising implementations of secure boot built upon the RISC-V ISA. They demonstrate minimal impact on system performance and offer additional security features, such as recovery or cryptography capabilities. However, it is vital to note that all these techniques require additional implementation on both hardware and software layers.

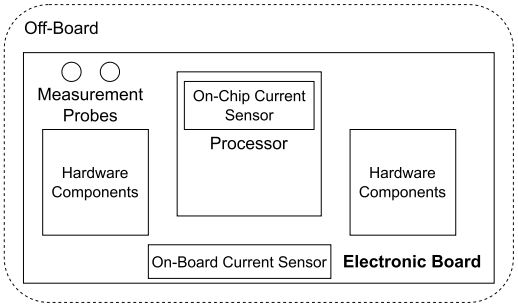

FIGURE 5. Ways of side-channels measurement.

Moving away from these active implementations, we will explore techniques that utilize existing internal indicators (passive implementations), like hardware performance counters or on-chip power sensors. We will also discuss techniques that leverage external power and electromagnetic field measurements, focusing on proposing a range of passive solutions.

B. ON-BOARD POWER MEASUREMENTS

On-board power measurement refers to measuring the power consumption of the device's motherboard. Power consumption has become a critical metric in various electronic devices for modern and critical applications [37]. It is possible to divide the power consumption into two main components:

- Static power: when the circuit is powered but not active, the consumption is due to transistor size reduction (gate-oxide current leakage).
- Dynamic power: when the circuit is active, the consumption is due to the switching activity of the transistors.

The dynamic part of the power consumption represents the activity in the circuit. Attackers can use this information to find secrets about the system. This is known as power side-channel attacks. However, this power can leak information that can be used for malware detection [38].

Recently, most electronic devices have come equipped with current or power sensors that dynamically measure power consumption. Such real-time measurements have become a key metric for assessing the energy efficiency of these devices. Currently, these measurements can serve as performance monitors, guiding decisions in the event of anomalies or potential malware activities.

In the study by [39], two primary techniques for power consumption measurement are discussed: on-board/on-chip and external measurements. As depicted in Figure 5, the on-board / on-chip approach takes advantage of the inherent power measurement capabilities of the hardware provided by the manufacturer. This method typically involves current sensors and voltage regulators. However, the accuracy of these measurements can be compromised due to inherent hardware noise, limitations in sampling frequency, and the low bit resolution of internal analog-to-digital converters.

However, external measurement techniques, which employ an external shunt resistor and external instrumentation, offer

improved accuracy and flexibility. This superior performance is primarily attributed to the reliance on external hardware. However, this method may not be suitable for on-board malware detection because it requires physical access and proximity to the power consumption measurement instruments.

In [38], the authors hypothesize that a malware execution produces enough changes in power usage to be able to detect its presence. To do so, they collected power profile data from a machine running in a clean state: only the OS, Internet Explorer (I.E.,) running, and Regedit running. The machine is then infected with a particular malware, and the power profiles of the machine running with the malware, the machine running Internet Explorer with the malware, and the machine running Regedit with the malware are collected. Therefore, six power profiles (three clean, three with the malware) are collected for each malware sample. Five malware samples are tested, all rootkits: Alureon, Pihar, Sirefef, Xpaj, and MaxRootkit.

Two approaches are tested for the data analysis:

- An unsupervised approach using ensemble voting of one-class anomaly detectors.
- A supervised approach using kernel-based Support Vector Machine (SVM).

The first approach uses several single-feature anomaly detectors that vote for the presence or absence of an anomaly. A value that deviates more than one standard deviation from the mean is flagged as an anomaly. The standard deviations and means are calculated from the training observations. This method reveals that the overall classifier achieves 100% True Detection Rate (TDR), with a lower False Detection Rate (FDR) than the supervised classifiers. The benchmark comprises 15 malicious power profiles (5 rootkits x 3 profiles) and 5 benign power profiles. Only one benign instance was detected as malicious (FDR $= 1/16$). The supervised approach shows a nearly perfect TDR and $17 - 18\%$ FDR.

The results confirm that rootkits alter power consumption and that those alterations can be detected. However, the results are obtained in an isolated test environment. Further work is required to test the robustness of this solution by adding noise and using different applications running concurrently. However, power profiling is a promising way to detect malware.

Other studies show how side channels can detect malware [40], [41], even in the presence of obfuscating techniques. A Convolution Neural Network (CNN) model trained with an electromagnetic trace is shown to have good accuracy. The same approach can be used with power trace analysis. Although the proposed techniques provide good results, they require external hardware and physical proximity. They are expensive in space, power, and cost, making them unsuitable in constrained environments and less likely to be widely developed.

C. ON-CHIP PERFORMANCE INDICATORS

On-chip performance indicators consist of all sensors and performance counters implemented inside the microprocessor. On-chip information can be power or performance indicators. Hardware indicators, such as hardware performance counters (HPC), can be used to detect malware execution on chips. Nowadays, software developers can rely on HPC to monitor the run-time performances of their programs on the target device. HPC are special purpose registers built into the Performance Monitoring Unit (PMU) of modern processors, which stores information about hardware events such as cache misses for data and instructions, branch prediction, load and store instruction, etc. [42]. HPC can be used for on-chip malware detection [43] because of the changes in the system's run-time behavior due to the malware execution. The authors in [44] have collected HPC data produced by malware and benign programs to train a set of classifiers to detect malware. This experiment has been carried out to detect Android malware and Linux rootkits. The study concludes that HPC can detect malware with relatively good accuracy. This study collects data through a Linux kernel module, which interrupts the execution every N cycle to retrieve information from the HPC. These methods slow down the system's performance and imply software-based security. That is why hardware implementation of such security features is preferred.

Malware Aware Processors (MAP), as presented in [45] and [46], provide real-time malware detection on the chip; they are implemented on hardware by using simple classifiers. MAP is presented as a first line of defense so that when malware is detected, heavier software can be triggered for malware detection and remediation. MAP uses low-level information that can be gathered and processed in hardware as:

- *Features related to architectural events*: use HPC to capture the number of memory reads, memory writes, unaligned memory accesses, immediate branches, and taken branches. The value of the features is collected for every 10,000 committed instructions. The collection of these features is called **ARCH**.
- *Features related to memory addresses*: measure the distance between the current load/store instruction memory address and the first load/store operation in the group of 10,000 instructions. The goal is to obtain a memory access pattern. Here, two collections are done: **MEM1** is the frequency of memory address distance histogram, and **MEM2** is the memory address distance histogram mix.
- *Features related to instruction*: look at the opcode and the instruction categories to get the frequency of usage of certain instruction categories or instruction opcodes. Four collections are done here, **INS1** is the frequency of instruction categories, **INS2** is the frequency of opcodes with the most significant difference, **INS3** is

TABLE 1. Comparison of different feature sets [46].

Feature	*LR (TP/FP)*		*NN (TP/FP)*	
Architectural events (ARCH)	70%	10%	88%	20%
Memory (MEM1)	90%	—	90%	4%
Instruction (INS2)	100%	16%	100%	9%

the existence of categories, and **INS4** is the existence of opcodes.

To know if malware is present with architectural events in each period, a list of opcodes showing the most significant difference in frequency between malware and regular programs has been established for instructions. In [47], a statistical analysis has been performed on the instructions used by the malware. For example, a botnet will show an increase in the frequency of the usage of the *mov* instruction and a decrease in the frequency of the use of the *push* instruction.

Two classification algorithms have been developed for MAP: a Logistic Regression (LR) algorithm that linearly separates regular programs from malware. The other model is a Neural Network (NN) made with perceptrons that approximate a classification function when trained. The results with True positive (TP) and false positive (FP) are presented in Table 1.

By combining all these features, a good enough detection can be achieved. However, it may increase the complexity of the MAP implementation significantly. In real-time detection, the problem should be considered as a time series analysis, where the time series is the successive decisions of the classifier. An Exponentially Weighted Moving Average is proposed to give more weight to recent inputs in [45].

Regarding the results, the processor has been implemented on FPGA using the INS2 feature with an LR prediction unit, as it is the most lightweight implementation in terms of complexity and impact on core power and area. This implementation shows an online detection accuracy of 100% with 16% false positives. Due to common false positives, MAP should be considered as a first line of defense that will trigger more resource-expensive solutions when a potential threat is discovered. While it is accurate to state that MAP offers effective lightweight malware detection on-chip, its usage necessitates the incorporation of an additional detector, possibly with boosted capabilities, to validate MAP's decisions due to the prevalence of false positives.

D. OFF-BOARD SIDE CHANNELS

Some techniques can help detect malware execution at runtime through off-board measurements. Off-board measurements consist of physical measurements performed from outside the main circuit board. In [40], the objective is to analyze the electromagnetic emanation of a device running software and to detect and classify malware. Here, the assumption that malware changes the device's behavior regarding electromagnetic emanation has already been proven.

The experiment used a Raspberry Pi 2B as the target device. This device is configured so that each malware execution will find valid files and will target them as it would normally. The malware dataset under test consists of DDoS malware: Mirai and Bashlite (a C&C server is present in the test environment to send commands to the infected machine), ransomware: Gonnarcy and kernel rootkits: Keysniffer, maK_It.

Electromagnetic emanations are acquired through an oscilloscope connected to an H-Field probe with an amplifier. In total, 3000 traces for each of the 30 malware binaries and 10,000 traces of benign activities are collected, making a dataset of 100,000 labeled traces.

For classification, neural networks are implemented because of their pattern recognition capabilities. Two architectures are compared: a Multi-Layer Perceptron (MLP) and a CNN. Other architectures, such as SVM or Naive Bayes, have also been tested but rejected due to their lack of robustness in more complex scenarios. The MLP takes flattened spectrogram bandwidths as input. The CNN comprises three blocks, each with one or two convolution layers followed by a max pooling layer. Here are the results:

- *Type classification*: Samples are categorized into four types: rootkit, ransomware, DDoS, and benign. All models show impressive performance in this category, achieving more accuracy than 98%. The CNN model stands out with the highest accuracy of 99.82%.
- *Family classification*: Samples are differentiated among various malware families in addition to the benign class. The results remain commendable, with an accuracy exceeding 97%. Once again, the CNN model achieves the highest accuracy at 99.61%.
- *Novelty classification*: Given the prevalence of variations of known malware in real-world scenarios, the models are evaluated against variations of the families being tested. These variations were not part of the training dataset. Despite this challenge, all models maintain a commendable accuracy of more than 92%. The CNN model shines with an accuracy of 99.38%.
- *Virtualization and packer identification*: The goal here is to determine whether the malware binary is shielded by virtualization or packing. This is achieved by comparing malware traces with traces of its protected variant, resulting in two distinct two-class detection challenges. The CNN model continues to lead with an accuracy of 95.83% for virtualization and 94.96% for packing.
- *Obfuscation classification*: In the final test, models are evaluated on their ability to classify seven obfuscation techniques: opaque predicates, bogus control flow, control flow flattening (either with O-LLVM or Tigress), instruction substitution, virtualization, and packing. The CNN model remains the front-runner, with an accuracy of 82.70%.

These results demonstrate that malware can be classified based on the electromagnetic emanations from the device under test, even when obfuscation techniques are employed to conceal the malware or when variations of known malware are introduced. The primary limitation of this method is the need for external devices equipped with probes and the substantial resources required to operate CNN or MLP. In particular, this solution may be more suitable for cloud and edge computing systems than embedded IoT and industrial IoT systems.

In the literature, notable efforts employ AI to detect malware based on processor information [44], [46] and side channels [38], [40]. Yet, a discernible gap exists: the elucidation of the underlying decision-making processes of these techniques still needs to be addressed.

This omission is particularly salient in the context of embedded IoT systems. While the efficacy of an AI model in detecting threats is undeniably valuable, the interpretability of its decisions is of equal, if not greater, importance. This brings us to the forefront of explainable AI (XAI).

XAI aims to demystify the intricate mechanisms of AI models, making them transparent and interpretable. Within the ambit of malware detection in embedded IoT, XAI can illuminate the specific attributes or patterns a model deems indicative of malicious activity. Such insights foster greater confidence in AI-driven solutions and equip cybersecurity researchers with nuanced understandings, paving the way for more robust and informed defense strategies. As we continue this discourse, the imperative of weaving XAI into our malware detection paradigms for embedded IoT becomes increasingly evident.

IV. EXPLAINABLE MACHINE LEARNING MODELS FOR MALWARE DETECTION

We divide the work on explainable machine learning for malware detection into two categories: software-assisted and hardware-assisted.

A. SOFTWARE-ASSISTED XAI FOR MALWARE DETECTION

MalConv [48] is a popular neural network designed for static analysis. It can be trained to analyze the bytes of a file to determine if it contains malware or goodware. This architecture is effective for detecting sequence patterns thanks to embedding and convolution layers, as shown in Figure 6. MalConv was tested for multi-class malware classification in [49].

The explainability of MalConv has been addressed in [50]. One of the methods used was decision boundary analysis. In a data space, that is, a space with enough dimensions to represent the possible combinations of the inputs, a decision boundary is a separation between data classes. In [50], the decision boundary is characterized by interpolating classified samples to find the moment of crossing from one class to another. This analysis is applied to the EmberMalConv network, which shares the same architecture as the Malconv network [48]. The results show that the crossing is

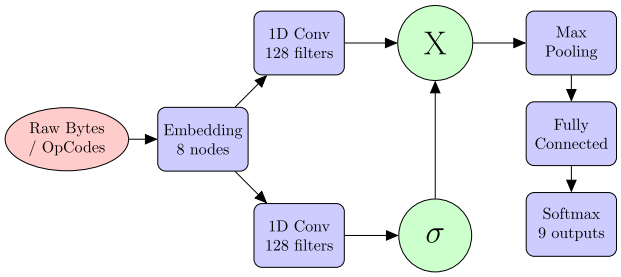

FIGURE 6. Malconv model for multi-class malware.

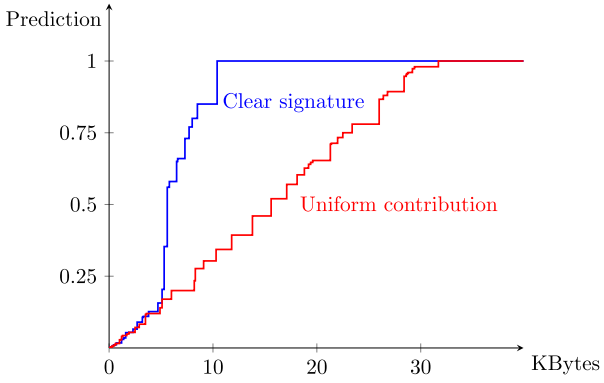

FIGURE 7. Decision Boundary Analysis (Smooth transitions indicate a uniform contribution of file contents. Abrupt transitions indicate a clear signature).

almost linear to the file modification, similar to the curve labeled "uniform contribution" in Figure 7. This linearity is considered a good sign of the robustness of the classifier, since no one-byte or few-byte abrupt changes can flip the decision. Furthermore, this method is considered a black-box method since it only requires access to the network's input and prediction (output layer).

The authors also used gradient analysis as an explainability technique, considered a white-box technique. During the training of a neural network, the weights at different layers are optimized using back-propagation. Back-propagation is merely an implementation of the chain rule for computing the gradient of the loss function. The loss function is specifically designed to align the network with the goal of the learning task.

By sorting the computed gradient magnitudes with respect to a test point, we can retrieve the most relevant features at intermediate stages and the input layer. The more the gradient value of a feature is important, the more it is relevant to the prediction.

This analysis was performed on the EmberMalConv network in [50] and the Malconv network in [51]. Both studies analyze variants of gradient-based techniques. They aim to find the parts of the file that the network mainly uses to determine maliciousness.

In the case of [50], the gradient analysis shows that there are definite spikes in the first block which contains the header, but also in other blocks in the input files. In [51], the gradient

analysis reaches a similar conclusion: the network tends to learn to discriminate between benign and malware samples based on the characteristics found in the file header. However, the actual maliciousness can be placed in executable files' data and text sections. Therefore, explainability gives us an idea of what the network has learned and how much we can trust it for real applications. If we ignore explainability, it is not unlikely that attackers can leverage it. For example, gradient analysis was used to design new adversarial attacks specifically tailored to a target network, as shown in [51].

FireEye [52] uses a similar deep learning network with embedding and convolution layers. For explainability, it extracts the weights of the activations in the network's first convolutional layer and uses them as low-level features. SHAP explainability produces a list of the most important low-level features. These features are then connected to their original semantics through parsing and disassembling the binary to arrive at human-understandable features.

In parallel, the reverse engineering of the file produces a list of interesting indicators of maliciousness called Areas of Interest (AOI). Finally, the authors examine the overlap between the "manual" reverse engineering results and the explainability of low-level features and provide expert analysis insights.

Increasingly, large language models are used for malware. For instance, IATelligence[2] brings information from GPT3 [53] to provide details about parts of the examined file. More specifically, GPT3 is used to analyze the Import Address Table of a Windows executable file and to determine if there are potential related MITRE ATT&CK[3] techniques that can be used against these APIs. It would be interesting to see if the LLM can be integrated into explainability pipelines for malware detection.

B. HARDWARE-ASSISTED XAI FOR MALWARE DETECTION

A few works on hardware-assisted malware detection using time series consider explainability. More efforts should be made in this area.

Pan et al. [12] propose a Recurring Neural Network model (RNN) for malware detection using embedded trace buffers (ETB) and HPC. Although their classifier is based on a multidimensional input signal, their explainability algorithm is unidimensional. To explain an input signal $\mathbf{w}$:

- Take the signal measurement at subsequent clock cycles of length d: $w_1, w_2, \ldots, w_d$.
- Perturb the measurement vector in n different ways to obtain n neighbor points.
- Obtain the output values by running the perturbations n through the RNN network.
- The perturbations and their outputs form a mini-dataset, which is used to train a linear regression model. The linear model mimics the RNN in the vicinity of the input signal $\mathbf{w}$.
- The linear model coefficients represent the weights with which the clock cycles contribute to the output value of the RNN. For example, the output value is the probability of a malware prediction. Therefore, the clock cycles with the highest weights are considered the explanation for the malicious behavior. By fetching the instructions executed at these cycles, a better idea of the program functionality is obtained as an explanation.

This method suffers from numerical issues since the subsequent clock cycles are strongly correlated, which leads to a numerically ill-conditioned problem. The authors address these issues by regularizing the regression model coefficients and limiting the number of perturbations. Another drawback of this method is that it tries to capture a linear regression model for a time series that can be rather long. While RNN is known to handle very long sequences, linear regression models are usually bad at this same task. The paper does not mention the typical length of the sequences used in experimentation. More importantly, the explanation is unidimensional, so we cannot capture the interaction between parallel signals.

LEMNA [17] uses a similar approach for explaining the detection of the start of a function in binary reverse engineering. The signal here is the sequence of bytes run through an RNN. Each byte is labeled as the start of a function or not. The LEMNA proposed explanation is different in two aspects:

- It is based on a mixture regression model.
- Its regularization penalizes the disparity between subsequent coefficients. It better captures the correlation between contiguous elements in the sequence.

The two methods deserve a comparative evaluation for tasks: malware classification and start-of-function detection; however, a method that captures multidimensional signals where an explanation may span different signals at different timestamps is still missing. Scalability to very long sequences is another challenge. Questions of how many perturbation points are required and what is an efficient method to sample these perturbations need to be sufficiently investigated.

V. DISCUSSION

In this section, we compare the different techniques presented in the article in terms of metrics for evaluating malware detection. Our survey reveals that hardware-based techniques show great accuracy, with low performance overhead compared to other solutions such as TPMs. These techniques are well suited for IoT and embedded devices.

Analyzing the current literature on detecting embedded malware, we have determined that ITUS [21] and CARE [20] effectively ensure the device's trustworthiness before usage. Additionally, CARE offers a recovery mechanism to avoid re-flashing the device in case of compromise. However, ITUS has more security modules, enabling the implementation of a Trusted Execution Environment (TEE) and enclaves.

[2]https://github.com/fr0gger/IATelligence
[3]https://attack.mitre.org

Other solutions require external hardware to exploit side channels to detect malware. These solutions show excellent results with near-perfect accuracy in malware detection and can even classify them even if obfuscating methods are in use [40]. These techniques have excellent efficiency; however, they require costly materials in terms of area, power consumption, and cost, making them difficult to implement.

In parallel, malware-aware processors [45], [46] are discussed. These solutions present efficient and lightweight on-chip malware detection at run-time and when the device is online. However, this solution can be considered a first line of defense, as it relies on precise and noise-free hardware performance counters on the chip. To this end, it is possible to summarize the related work in Table 2. For this purpose, we define the following features for each approach as follows:

- **Secure Boot**: It implements secure boot.
- **Detection at run-time**: It detects malware at run-time.
- **Byte analysis**: It uses the binary file to detect malware.
- **Recovery**: It recovers from a malware infection.
- **AI-based**: It uses artificial intelligence techniques for malware detection.
- **Hardware-based**: It uses hardware components for malware detection.
- **Malware classification**: It can classify the malware types.
- **External device required**: It requires an additional system for malware detection.

In the context of IoT, malware detection using side channels takes work. This is due to the fact that the implementation cost is very high per single device. To this end, it is possible to use side channels data for malware detection using external instrumentation to check a specific IoT node at run-time. This solution is unsuitable for numerous IoT devices in a network, such as thousands of nodes.

MAP (Malware Aware Processor) is very promising for mitigating malware execution. Indeed, MAP aims to include additional hardware counters that feed performance data to detector models to help them decide whether malware is executed. This solution is scalable and portable, as it can be implemented on different architectures such as RISC-V, x86, and many others. Moreover, this solution is lightweight and easy to implement; thus, it is suitable for low-cost and power-limited IoT devices.

Hardware-based malware detection and prevention has the potential to improve computer security by identifying and mitigating threats at a lower level within the hardware. However, this approach also presents several challenges. One of the primary challenges is that hardware must be designed with security in mind, which requires significant resources and expertise. Furthermore, hardware-assisted malware detection and prevention has limitations. For example, with HPC, in particular, when several programs are executed [54], it becomes difficult to track the behavior of individual programs.

Furthermore, an HPC malware detector can affect the overall system performance [20], [21]. For example, using the MAP solution [45], this can lead to an increase of up to 13.12% in area, 5.23% in power consumption, and a loss of 2.28% in terms of maximum operating frequency, and this only by simply using a basic neural network model. Finding simpler models and better implementations is crucial, as the trade-off between security and performance must be considered during the design phase. An alternative security model based on logistic regression is proposed [45]. In this work, the implementation of the model has an impact of an increase of 0.28% in terms of logic cells, an increase of 0.08% in power usage and a loss of 1.93% in terms of operating frequency. Implementing an efficient hardware malware detector has consequences on the performance of the system in terms of power, frequency, and area. The trade-off between security and performance has to be considered directly at design time. More sophisticated models, such as neural networks, may significantly impact devices and may not be suitable for certain IoT applications where power consumption and area are critical.

In parallel, we propose a set of dimensions to compare malware detection techniques. These dimensions are summarized as follows:

- **Detection accuracy**: This is arguably the most critical metric. It quantifies the ability of a detection system to identify malware samples correctly. The metrics under this category include True Positive Rate (TPR), which represents the proportion of actual malware samples that are correctly identified. False Positive Rate (FPR) denotes the proportion of benign samples mistakenly identified as malware. A high FPR can be detrimental, leading to unnecessary interventions or disruptions. The Receiver Operating Characteristic (ROC) curve is a graphical representation that illustrates the performance of a binary classifier system as its discrimination threshold is varied. It is a plot of the TPR against the FPR.
- **Performance Overhead**: This metric measures the impact of the malware detection technique on overall system performance. It is typically measured in terms of CPU cycles or time. A detection system with high performance overhead might not be feasible for real-time applications.
- **Scalability**: As the name suggests, this metric assesses the ability of the detection system to handle large-scale malware detection tasks. Factors like system resource requirements, limitations of detection rates at scale, and detection speed play a crucial role in determining scalability.
- **Robustness**: A robust malware detection system should be resilient against adversaries' evasion and obfuscation techniques. This metric evaluates the system's ability to detect zero-day exploits and its resilience to various attacks.

The table 3 provides an overview of state-of-the-art hardware-assisted malware detection techniques. Various

TABLE 2. Categorization of discussed related work.

Method	Secure boot	Detection run-time	Byte analysis	Recovery	AI based	Hardware based	Malware class.	External device
CARE [20]	✓			✓		✓		
ITUS [21]	✓					✓		
HPCs [44]		✓			✓	✓	✓	
MAP [45], [46]		✓			✓	✓		
Energy Prof. [38]		✓			✓			✓
Elec. field [40]		✓			✓		✓	✓
TPM [33]	✓	✓				✓		
DICE [36]	✓					✓		
Malconv [50]			✓		✓			
Fireye [52]			✓		✓			
LEMNA [17]					✓		✓	

techniques exhibit distinct strengths and limitations. For instance, CARE and ITUS stand out for their high detection accuracy during boot time. However, they are accompanied by moderate performance overhead, and their limited scalability and robustness suggest potential constraints in their applicability, particularly in large-scale deployments or scenarios plagued by sophisticated attacks. On the other hand, techniques rooted in the electromagnetic field and power consumption patterns showcase exemplary detection accuracy at run-time. Their low performance overhead positions them as prime candidates for real-time applications, and their notable robustness underscores their resilience against advanced evasion stratagems. MAP and HPCs present a balanced profile with moderate detection accuracy at run-time and a performance overhead that ranges from moderate to low, hinting at their efficacy in more controlled environments. TPM emerges as a robust technique, offering high boot-time detection accuracy, albeit at the cost of a high-performance overhead. However, its impressive scalability and robustness metrics herald its potential for expansive deployments. Similarly, DICE mirrors TPM in its boot-time detection, boasts a low performance overhead. While its scalability is laudable, the absence of specified robustness metrics necessitates further exploration. Lastly, LEMNA distinguishes itself with its run-time detection ability. Although its performance overhead remains unspecified, its moderate scalability hints at its suitability for medium- to large-scale operational environments.

In conclusion, the landscape of hardware-assisted malware detection is diverse, with each technique having strengths and limitations. The choice of technique largely depends on the specific requirements of the deployment scenario, such as real-time constraints, scale, and the sophistication of potential threats. As malware continues to evolve, it is imperative for detection techniques to adapt and innovate, ensuring safer digital systems.

VI. CHALLENGES AND OPPORTUNITIES

Besides explainability, we discuss future directions of malware detection for modern processors, their challenges, and opportunities.

A. ADVERSARIAL TRAINING

As malware tactics evolve and adapt, traditional hardware-based solutions may become less effective and require continuous investment in hardware design and updates. The issue of malware, particularly ransomware, has become highly profitable. In this report [55], various types of ransomware are presented, each with its implementation strategy depending on the intended target and the method of spreading. This has created an actual arms race between malware authors and security providers. Initially, malware detectors scanned executables for known malicious instructions, but malware authors encrypted their payload in response. As detectors began looking for decryption codes, malware authors responded by randomly changing their payload using different compilation strategies [44]. Today, detectors rely on observing the behavior of execution, and, as a result, malware authors are focusing on obfuscation techniques to conceal the presence of their malware. Binary manipulation is effective in evading state-of-the-art detection engines. Four state-of-the-art malware detection models, together with five industry-standard malware detection engines, have been analyzed in [56] to find that these models are susceptible to various binary manipulation-based threats, including binary padding and section injection attacks. The authors propose to reduce the attack surface by uncovering the sensitivity of volatile features within the detection engines and exhibiting their exploitability. A graph-based malware detection scheme is proposed to eliminate volatile information. For these reasons, adversarial training for raw-binary malware classifiers is gaining traction [57].

B. ZERO-TRUST ARCHITECTURES

These architectures [58] assume that no system component is trusted by default. Verifying every action, communication, and access request helps prevent lateral movements of malware.

C. SUPPLY CHAIN SECURITY

Moreover, securing the supply chain from software development to hardware manufacturing mitigates the presence of

TABLE 3. Comparison of merits of state of the art hardware assisted malware detection.

Technique	Detection Accuracy (Boot/Run-time)	Performance Overhead	Scalability	Robustness
CARE [20] & ITUS [21]	High/-	Moderate	Low	Low
Electromagnetic field [40]	-/High	Low	-	High
Power Consumption [38]	-/High	Low	-	High
MAP [46] & HPCs [44]	-/Moderate	Moderate	Low	-
TPM [33]	High/-	High	High	-
DICE [16]	High/-	Low	High	-
LEMNA [17]	-/High	-	Moderate	-

backdoors and vulnerabilities. Such techniques include code signing, secure boot processes, and firmware analysis [59].

D. HARDWARE-ASSISTED DETECTION

Although there are challenges, hardware-assisted malware detection and prevention presents substantial opportunities. By detecting malware at a deeper hardware level, these solutions can provide an extra layer of protection against both known and unknown threats. Moreover, hardware-based solutions can isolate different components of a system, making it harder for malware to propagate. Furthermore, hardware-assisted solutions can address some limitations of software-based approaches, including the inability to detect certain types of malware. Simple machine learning algorithms, such as logistic regression, in addition to malware-aware processors have been shown to have good accuracy with silicon small areas and low-power [46].

E. COLLABORATIVE SW/HW SOLUTIONS

Future work may focus on collaborative SW/HW malware detection, such as those presented in [5]. When working together, software and hardware components can offer a more comprehensive view of the system, allowing real-time identification and response to threats. Collaborative SW/HW malware detection provides a significant advantage in defending against sophisticated and targeted malware attacks. Such attacks frequently exploit vulnerabilities in both software and hardware components. Therefore, combining both software and hardware detection techniques can offer a more robust defense to identify and mitigate threats. In addition, collaborative SW/HW malware detection has the advantage of making more efficient use of system resources. By offloading some malware detection and prevention tasks to the hardware, the software can concentrate on other essential security tasks, thereby minimizing the performance overhead of software-based security solutions. The adaptability of SW/HW malware detection to address emerging threats is a significant advantage in such a fast-moving field.

F. OPERATING SYSTEM-INDEPENDENT DETECTION

Integrating OS information is undoubtedly beneficial when the device is running one. But if a model was trained on devices with no OS and is then used on a device with an OS, the results may be erroneous, as the model may consider the running OS to be malware. State-of-the-art hardware-assisted detection reveals that models are trained on specific operating systems, such as Windows [60] or Android [61], [62]. Current run-time detection models are architecture and operating system dependent, making it difficult to develop generic malware detectors. Therefore, one main challenge is developing operating system-independent malware detectors. For instance, HPC-based solutions can work with different operating systems. For example, the authors in [54] propose to use HPCs and the operating system for static (load time) and dynamic (runtime) integrity checking of programs. For an interesting take on the challenges, pitfalls, and perils of using HPC for security, we refer the reader to [63].

VII. CONCLUSION

Integrating hardware in the battle against malware represents a significant step forward, offering a robust layer of defense that complements software-based solutions. Secure boot architectures, malware-aware processors, and the utilization of hardware performance indicators underscore the potential of this integration. However, as we navigate the complexities of these approaches, particularly in the realm of embedded systems, the challenges, and opportunities become evident. While secure architectures form the bedrock of device protection, the potential of side-channel techniques, despite their integration challenges, should be considered.

This paper has presented various state-of-the-art techniques for detecting embedded malware and described the use of explainable machine learning methods. Side channels and passive methods are promising for dynamic analysis, and Malconv-like solutions are effective for static analysis.

In addition, we have explored various approaches to explainability, including gradient analysis, model surrogates, and decision boundary analysis. Explainability enables a better understanding of what a model has learned, helping a human analyst filter false positives and deter evasion tactics.

In future work on this topic, we aim to implement hardware side-channel methods and support them with a vertical explainability layer to connect predictions and patterns. This interdisciplinary research requires a collaborative effort among researchers from hardware security and AI experts.

REFERENCES

[1] Cisco. (2019). *Internet of Things at-a-Glance*. [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/se/internet-of-things/at-a-glance-c45-731471.pdf

[2] Md. M. Hossain, M. Fotouhi, and R. Hasan, "Towards an analysis of security issues, challenges, and open problems in the Internet of Things," in *Proc. IEEE World Congr. Services*, Jun. 2015, pp. 21–28.

[3] R. L. Castro, C. Schmitt, and G. Dreo, "AIMED: Evolving malware with genetic programming to evade detection," in *Proc. 18th IEEE Int. Conf. Trust, Secur. Privacy Comput. Commun./13th IEEE Int. Conf. Big Data Sci. Eng.*, Aug. 2019, pp. 240–247.

[4] Z. Pan, J. Sheldon, C. Sudusinghe, S. Charles, and P. Mishra, "Hardware-assisted malware detection using machine learning," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Feb. 2021, pp. 1775–1780.

[5] M. Botacin, M. Z. Alves, D. Oliveira, and A. Grégio, "HEAVEN: A hardware-enhanced AntiVirus ENgine to accelerate real-time, signature-based malware detection," *Expert Syst. Appl.*, vol. 201, Sep. 2022, Art. no. 117083.

[6] L. Zhou and Y. Makris, "Hardware-assisted rootkit detection via on-line statistical fingerprinting of process execution," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2018, pp. 1580–1585.

[7] *Sonicwall Cyber Threat Report*, SonicWall Inc., San Jose, CA, USA, 2022.

[8] Ö. A. Aslan and R. Samet, "A comprehensive review on malware detection approaches," *IEEE Access*, vol. 8, pp. 6249–6271, 2020.

[9] P. Mell, K. Kent, and J. Nusbaum, "Guide to malware incident prevention and handling," U.S. Dept. Commerce, Technol. Admin., Nat. Inst. Standards Technol. (NIST), Gaithersburg, MD, USA, Tech. Rep. 800-83, 2005. [Online]. Available: https://www.nist.gov/publications/guide-malware-incident-prevention-and-handling

[10] L. R. Bays, R. R. Oliveira, M. P. Barcellos, L. P. Gaspary, and E. R. M. Madeira, "Virtual network security: Threats, countermeasures, and challenges," *J. Internet Services Appl.*, vol. 6, no. 1, pp. 1–19, Dec. 2015.

[11] G. Ras, N. Xie, M. Van Gerven, and D. Doran, "Explainable deep learning: A field guide for the uninitiated," *J. Artif. Intell. Res.*, vol. 73, pp. 329–397, Jan. 2022.

[12] Z. Pan, J. Sheldon, and P. Mishra, "Hardware-assisted malware detection using explainable machine learning," in *Proc. IEEE 38th Int. Conf. Comput. Design (ICCD)*, Oct. 2020, pp. 663–666.

[13] M. T. Ribeiro, S. Singh, and C. Guestrin, "Anchors: High-precision model-agnostic explanations," in *Proc. AAAI Conf. Artif. Intell.*, vol. 32, 2018, pp. 1–20.

[14] M. T. Keane and B. Smyth, "Good counterfactuals and where to find them: A case-based technique for generating counterfactuals for explainable AI (XAI)," in *Case-Based Reasoning Research and Development*. Cham, Switzerland: Springer, 2020, pp. 163–178.

[15] M. T. Ribeiro, S. Singh, and C. Guestrin, "'Why should I trust you?': Explaining the predictions of any classifier," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2016, pp. 1135–1144.

[16] J. E. Zini, M. Mansour, and M. Awad, "CEnt: An entropy-based model-agnostic explainability framework to contrast classifiers' decisions," 2023, *arXiv:2301.07941*.

[17] W. Guo, D. Mu, J. Xu, P. Su, G. Wang, and X. Xing, "LEMNA: Explaining deep learning based security applications," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2018, pp. 364–379.

[18] J. Butterworth, C. Kallenberg, X. Kovah, and A. Herzog, "BIOS chronomancy: Fixing the core root of trust for measurement," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, New York, NY, USA, 2013, pp. 25–36.

[19] Microsoft. (2023). *Secure Boot*. Accessed: Jul. 20, 2023. [Online]. Available: https://learn.microsoft.com/en-us/windows-hardware/design/device-experiences/oem-secure-boot

[20] A. Dave, N. Banerjee, and C. Patel, "CARE: Lightweight attack resilient secure boot architecture with onboard recovery for RISC-V based SOC," in *Proc. 22nd Int. Symp. Quality Electron. Design (ISQED)*, Apr. 2021, pp. 516–521.

[21] V. B. Y. Kumar, A. Chattopadhyay, J. Haj-Yahya, and A. Mendelson, "ITUS: A secure RISC-V system-on-chip," in *Proc. 32nd IEEE Int. Syst.-Chip Conf. (SOCC)*, Sep. 2019, pp. 418–423.

[22] R. Coombs, "Securing the future of authentication with ARM TrustZone-based trusted execution environment and fast identity online (FIDO)," ARM, White Paper, 2015.

[23] *A Technical Analysis of Confidential Computing V1.1*, Confidential Computing Consortium, San Francisco, CA, USA, 2021.

[24] M. T. Aga and T. Austin, "Smokestack: Thwarting DOP attacks with runtime stack layout randomization," in *Proc. IEEE/ACM Int. Symp. Code Gener. Optim. (CGO)*, Feb. 2019, pp. 26–36.

[25] P. Borrello, D. C. D'Elia, L. Querzoni, and C. Giuffrida, "Constantine: Automatic side-channel resistance using efficient control and data flow linearization," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, New York, NY, USA, Nov. 2021, pp. 715–733.

[26] Z. Chang, D. Xie, and F. Li, "Oblivious RAM: A dissection and experimental evaluation," *Proc. VLDB Endowment*, vol. 9, no. 12, pp. 1113–1124, Aug. 2016.

[27] X. Li, Y. Wen, M. H. Huang, and Q. Liu, "An overview of bootkit attacking approaches," in *Proc. 7th Int. Conf. Mobile Ad-Hoc Sensor Netw.*, Dec. 2011, pp. 428–431.

[28] J. Hendricks and L. van Doorn, "Secure bootstrap is not enough: Shoring up the trusted computing base," in *Proc. 11th Workshop ACM SIGOPS Eur. Workshop*, Sep. 2004, p. 11.

[29] T. M. Chen and S. Abu-Nimeh, "Lessons from Stuxnet," *Computer*, vol. 44, no. 4, pp. 91–93, Apr. 2011.

[30] S. Collins and S. McCombie, "Stuxnet: The emergence of a new cyber weapon and its implications," *J. Policing, Intell. Counter Terrorism*, vol. 7, no. 1, pp. 80–91, Apr. 2012.

[31] M. S. Haghighi, F. Farivar, A. Jolfaei, A. B. Asl, and W. Zhou, "Cyber attacks via consumer electronics: Studying the threat of covert malware in smart and autonomous vehicles," *IEEE Trans. Consum. Electron.*, early access, Aug. 2, 2023, doi: 10.1109/TCE.2023.3297965.

[32] C. Profentzas, M. Günes, Y. Nikolakopoulos, O. Landsiedel, and M. Almgren, "Performance of secure boot in embedded systems," in *Proc. 15th Int. Conf. Distrib. Comput. Sensor Syst. (DCOSS)*, May 2019, pp. 198–204.

[33] *Trusted Platform Module (TPM) Summary*, Trusted Comput. Group Admin., Beaverton, OR, USA, 2008.

[34] Z. Ling, H. Yan, X. Shao, J. Luo, Y. Xu, B. Pearson, and X. Fu, "Secure boot, trusted boot and remote attestation for ARM TrustZone-based IoT nodes," *J. Syst. Archit.*, vol. 119, Oct. 2021, Art. no. 102240.

[35] M. Sidheeq, A. Dehghantanha, and G. Kananparan, "Utilizing trusted platform module to mitigate BotNet attacks," in *Proc. Int. Conf. Comput. Appl. Ind. Electron.*, Dec. 2010, pp. 245–249.

[36] Z. Tao, A. Rastogi, N. Gupta, K. Vaswani, and A. V. Thakur, "DICE: A formally verified implementation of DICE measured boot," in *Proc. 30th USENIX Secur. Symp.*, Aug. 2021, pp. 1091–1107.

[37] Y. Nasser, C. Sau, J.-C. Prévotet, T. Fanni, F. Palumbo, M. Hélard, and L. Raffo, "NeuPow: A CAD methodology for high-level power estimation based on machine learning," *ACM Trans. Design Autom. Electron. Syst.*, vol. 25, no. 5, pp. 1–29, Aug. 2020.

[38] R. Bridges, J. Hernández Jiménez, J. Nichols, K. Goseva-Popstojanova, and S. Prowell, "Towards malware detection via CPU power consumption: Data collection design and analytics," in *Proc. 17th IEEE Int. Conf. Trust, Secur. Privacy Comput. Commun./12th IEEE Int. Conf. Big Data Sci. Eng.*, Aug. 2018, pp. 1680–1684.

[39] Y. Nasser, J. Lorandel, J.-C. Prévotet, and M. Hélard, "RTL to transistor level power modeling and estimation techniques for FPGA and ASIC: A survey," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 3, pp. 479–493, Mar. 2021.

[40] D.-P. Pham, D. Marion, M. Mastio, and A. Heuser, "Obfuscation revealed: Leveraging electromagnetic signals for obfuscated malware classification," in *Proc. Annu. Comput. Secur. Appl. Conf.*, Dec. 2021, pp. 706–719.

[41] H. A. Khan, N. Sehatbakhsh, L. N. Nguyen, M. Prvulovic, and A. Zajić, "Malware detection in embedded systems using neural network model for electromagnetic side-channel signals," *J. Hardw. Syst. Secur.*, vol. 3, no. 4, pp. 305–318, Dec. 2019.

[42] J. Dongarra, K. London, S. Moore, P. Mucci, D. Terpstra, H. You, and M. Zhou, "Experiences and lessons learned with a portable interface to hardware performance counters," in *Proc. Int. Parallel Distrib. Process. Symp.*, 2003, p. 6, doi: 10.1109/IPDPS.2003.1213517.

[43] A. Tang, S. Sethumadhavan, and S. J. Stolfo, "Unsupervised anomaly-based malware detection using hardware features," in *Research in Attacks, Intrusions and Defenses*, A. Stavrou, H. Bos, and G. Portokalidis, Ed. Cham, Switzerland: Springer, 2014, pp. 109–129.

[44] 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 Comput. Archit. News*, vol. 41, no. 3, pp. 559–570, Jun. 2013.

[45] M. Ozsoy, K. N. Khasawneh, C. Donovick, I. Gorelik, N. Abu-Ghazaleh, and D. Ponomarev, "Hardware-based malware detection using low-level architectural features," *IEEE Trans. Comput.*, vol. 65, no. 11, pp. 3332–3344, Nov. 2016.
[46] M. Ozsoy, C. Donovick, I. Gorelik, N. Abu-Ghazaleh, and D. Ponomarev, "Malware-aware processors: A framework for efficient online malware detection," in *Proc. IEEE 21st Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2015, pp. 651–661.
[47] D. Bilar, "Opcodes as predictor for malware," *Int. J. Electron. Secur. Digit. Forensics*, vol. 1, no. 2, p. 156, 2007.
[48] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. K. Nicholas, "Malware detection by eating a whole exe," in *Proc. Workshops 32nd AAAI Conf. Artif. Intell.*, Oct. 2017.
[49] M. A. Kadri, M. Nassar, and H. Safa, "Transfer learning for malware multi-classification," in *Proc. 23rd Int. Database Appl. Eng. Symp.*, 2019, pp. 1–7.
[50] S. Bose, T. Barao, and X. Liu, "Explaining AI for malware detection: Analysis of mechanisms of MalConv," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2020, pp. 1–8.
[51] L. Demetrio, B. Biggio, G. Lagorio, F. Roli, and A. Armando, "Explaining vulnerabilities of deep learning to adversarial malware binaries," 2019, *arXiv:1901.03583*.
[52] S. E. Coull and C. Gardner, "Activation analysis of a byte-based deep neural network for malware classification," in *Proc. IEEE Secur. Privacy Workshops (SPW)*, May 2019, pp. 21–27.
[53] T. B. Brown et al., "Language models are few-shot learners," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, May 2020, pp. 1877–1901.
[54] C. Malone, M. Zahran, and R. Karri, "Are hardware performance counters a cost effective way for integrity checking of programs," in *Proc. 6th ACM Workshop Scalable Trusted Comput.*, Oct. 2011, pp. 71–76.
[55] *État de la Menace Rançongiciel*, ANSSI, Boulevard de La Tour-Maubourg, Paris, France, 2021.
[56] A. Abusnaina, Y. Wang, S. Arora, K. Wang, M. Christodorescu, and D. Mohaisen, "Burning the adversarial bridges: Robust windows malware detection against binary-level mutations," 2023, *arXiv:2310.03285*.
[57] K. Lucas, S. Pai, W. Lin, L. Bauer, M. K. Reiter, and M. Sharif, "Adversarial training for raw-binary malware classifiers," in *Proc. 32nd USENIX Secur. Symp.*, 2023, pp. 1163–1180.
[58] N. F. Syed, S. W. Shah, A. Shaghaghi, A. Anwar, Z. Baig, and R. Doss, "Zero trust architecture (ZTA): A comprehensive survey," *IEEE Access*, vol. 10, pp. 57143–57179, 2022.
[59] V. Hassija, V. Chamola, V. Gupta, S. Jain, and N. Guizani, "A survey on supply chain security: Application areas, security threats, and solution architectures," *IEEE Internet Things J.*, vol. 8, no. 8, pp. 6222–6246, Apr. 2021.
[60] D. Gibert, C. Mateu, and J. Planes, "The rise of machine learning for detection and classification of malware: Research developments, trends and challenges," *J. Netw. Comput. Appl.*, vol. 153, Mar. 2020, Art. no. 102526.
[61] B. Amos, H. Turner, and J. White, "Applying machine learning classifiers to dynamic Android malware detection at scale," in *Proc. 9th Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, Jul. 2013, pp. 1666–1671.
[62] K. Liu, S. Xu, G. Xu, M. Zhang, D. Sun, and H. Liu, "A review of Android malware detection approaches based on machine learning," *IEEE Access*, vol. 8, pp. 124579–124607, 2020.
[63] 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. IEEE Symp. Secur. Privacy (SP)*, May 2019, pp. 20–38.

YEHYA NASSER received the master's degree from Lebanese International University, Lebanon, now known as Beirut International University, in 2016, and the Ph.D. (European and French) degree from INSA Rennes, France, in 2019. He is currently a Researcher in hardware digital design and low-power secure embedded systems, boasting more than ten years of experience and a pronounced emphasis on hardware security. He is also an Associate Professor with the Engineering and Research Center, IMT Atlantique School, France. Prior to his tenure with IMT Atlantique School, he enriched his expertise during his research and development role with NOKIA, France. His academic roots trace back to Lebanese International University, Lebanon, now known as Beirut International University. Beyond the academic realm, he has left a significant mark on the industry. He was a Research and Development Engineer with NOKIA, from 2019 to 2021. His research interests include the domains of low-power, secure hardware, and embedded systems.

MOHAMAD NASSAR (Member, IEEE) received the master's degree (D.E.A.) in computer science and the Ph.D. degree from Nancy University (currently the University of Lorraine), France, in 2005 and 2009, respectively. He is currently a tenure-track Assistant Professor in computer science with The University of Alabama in Huntsville. He was an Assistant Professor in computer science and cybersecurity with the University of New Haven (UNewHaven) (2021–2023). Before that, he was an Assistant Professor in computer science with the American University of Beirut (AUB) (2016–2021). Before joining AUB, he completed a postdoctoral research stay with the Department of Computer Science and Engineering, Qatar University. He was an Expert Research Engineer with INRIA, Nancy, France (2009–2010) and Ericsson, Ireland (2011). He has published more than 40 peer-reviewed conference and journal articles. His research interests include cybersecurity and machine learning.