IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 2 February 2024, accepted 17 March 2024, date of publication 15 April 2024, date of current version 22 April 2024.

*Digital Object Identifier 10.1109/ACCESS.2024.3388716*

SURVEY

# A Survey on Hardware-Based Malware Detection Approaches

**CRISTIANO PEGORARO CHENET, (Student Member, IEEE), ALESSANDRO SAVINO, (Senior Member, IEEE), AND STEFANO DI CARLO, (Senior Member, IEEE)**
Department of Control and Computer Engineering, Politecnico di Torino, 10129 Turin, Italy

Corresponding author: Stefano Di Carlo (stefano.dicarlo@polito.it)

This work was supported in part by the Project SERICS through the MUR National Recovery and Resilience Plan funded by the European Union—NextGenerationEU under Grant PE00000014, and in part by the Vitamin-V Project funded by the European Union under Project 101093062.

**ABSTRACT** This paper delves into the dynamic landscape of computer security, where malware poses a paramount threat. Our focus is a riveting exploration of the recent and promising hardware-based malware detection approaches. Leveraging hardware performance counters and machine learning prowess, hardware-based malware detection approaches bring forth compelling advantages such as real-time detection, resilience to code variations, minimal performance overhead, protection disablement fortitude, and cost-effectiveness. Navigating through a generic hardware-based detection framework, we meticulously analyze the approach, unraveling the most common methods, algorithms, tools, and datasets that shape its contours. This survey is not only a resource for seasoned experts but also an inviting starting point for those venturing into the field of malware detection. However, challenges emerge in detecting malware based on hardware events. We struggle with the imperative of accuracy improvements and strategies to address the remaining classification errors. The discussion extends to crafting mixed hardware and software approaches for collaborative efficacy, essential enhancements in hardware monitoring units, and a better understanding of the correlation between hardware events and malware applications.

**INDEX TERMS** Cybersecurity, malware, hardware-based detection, hardware-based framework.

## I. INTRODUCTION

Malware, short for malicious software, poses a significant threat to computer security. It includes any code modification within a software system aimed at causing harm or disrupting the system's intended function [1], [2]. Malware attacks cover spying, intrusive ads, email abuse, system damage, ransom demands, data release, slowdown, browser manipulation, and unauthorized access to sensitive information. Successful attacks lead to consequences that can be categorized into four groups: (i) unauthorized disclosure, where an authorized entity gains access to data; (ii) deception, where an authorized entity receives false data; (iii) disruption, causing interruptions in system services; and (iv) usurpation, resulting in unauthorized control of system services [3]. Computing systems, including personal computers, mobile phones, Internet of Things (IoT), 5G devices, Cyber-Physical Systems (CPSs), and enterprise-wide systems, are vulnerable to malware. The complexity and size of modern systems, often indicated by a rising number of lines of code, amplify the threat. Factors such as numerous bugs, unsafe programming languages, improper configuration, and the ease of concealing malicious code create potential vulnerabilities. Additionally, the increased network connectivity expands the security risks, making all devices potential targets for attackers. For example, cybercrimes have seen a 70% increase in online fraud accomplished through mobile platforms, with a 30% rise in IoT malware in 2020 [4].

Globally, cybersecurity is paramount, with malware being a primary vehicle for cybercrimes. The World Economic

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

Forum Global Risk Report 2023 ranks cyber insecurity eighth among top global risks, alongside threats like climate change and involuntary migration [5]. Cybersecurity Ventures predicts a 15 percent annual growth in international cybercrime costs, reaching USD 8 trillion in 2023 and USD 10.5 trillion annually by 2025 [5]. Global spending on cybersecurity products and services is expected to exceed USD 1.75 trillion from 2021 to 2025, growing 15 percent year-over-year [6]. Ransomware, a prevalent malware threat, was predicted to cost USD 20 billion globally in 2021, with damage costs projected to exceed USD 265 billion annually by 2031 [5].

Researchers have developed various malware detection methods in response to these alarming statistics, leveraging Machine Learning (ML) and Deep Learning (DL) techniques. Surveys have evaluated and categorized research in this domain, focusing on specific Operating Systems (OSs), such as Windows, or mobile platforms like Android. Ye et al. [7] conducted a comprehensive survey on intelligent malware detection using data mining techniques, emphasizing the importance of Feature Extraction (FE) and algorithm selection. Subsequently, Ucci et al. [8] provided an overview of machine learning-based malware analysis, focusing on analysis objectives, FE, and ML algorithms, albeit limited to Portable Executable (PE) files. Gibert et al. [9] systematically reviewed ML and DL techniques for Windows malware detection, comparing input features, classification algorithms, and dataset characteristics. Similarly, Qiu et al. [10] and Liu et al. [11] addressed deep Android malware detection, emphasizing supervised classification using Multilayer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs) architectures. Catal et al. [12] conducted an extensive literature review on DL techniques for mobile malware detection, highlighting the prevalence of MLP and CNN architectures, with a focus on supervised learning and static features. Furthermore, Deldar and Abadi [13] proposed a survey on DL techniques for zero-day malware detection, targeting features extracted at the software level to address emerging threats.

In the early 2010s, researchers initially proposed the idea of Hardware-Supported Malware Detection (HMD) [14], [15]. HMD involves dynamically analyzing micro-architecture events in a processor using ML algorithms to differentiate between benign applications and malware. The shift towards HMD is justified because of the potential of enhanced security by leveraging robust hardware monitoring infrastructures. This provides a more robust defense against sophisticated attacks that may exploit vulnerabilities in software-based approaches. Specifically, hardware features reflect phase behavior in the underlying hardware, as observed in prior studies [16], [17]. These phases often correspond to time-behavioral patterns in micro-architectural events, which vary significantly between programs, enabling the distinction between malicious and benign applications. Additionally, these hardware-based approaches address the zero-day issue, as demonstrated in [18]. To the best of our knowledge, a comprehensive overview of HMD methods is still missing. This paper tries to cover this gap.

The structure is as follows: Section II covers the basics of malware, serving as a foundation for understanding the field. Section III presents a comprehensive overview of software and hardware-based malware detection solutions, with a detailed discussion of their strengths and weaknesses. Section IV delves into crucial aspects of hardware-based detection. Lastly, Section VI provides conclusions and outlines research challenges.

## II. MALWARE FUNDAMENTALS

Categorizing malware is difficult because of its growing complexity and diverse properties. Yet, creating a malware taxonomy provides valuable insights into understanding it better. Before exploring the fundamentals of malware operation, let us define a set of keywords commonly used to describe different malware categories [1], [19]:

- **Virus**: malicious code with the capability of inserting itself into other programs;
- **Worm**: malicious code that propagates similarly to viruses but does not require a target software to replicate, often exploiting connectivity such as emails;
- **Trojan horse**: malicious code that masquerades as a useful program;
- **Spyware**: malicious code secretly installed into an information system to transmit private user data to an external entity;
- **Adware**: malicious code that displays computer advertisements, primarily aiming for financial benefits;
- **Ransomware**: malicious code that denies access to a user's data, usually by encrypting it until a ransom is paid;
- **Backdoor**: malicious code that opens systems to external entities by subverting local security policies to allow remote access and control over a network;
- **Keylogger**: malicious code designed to record keystrokes, used to obtain passwords or encryption keys to bypass security measures;
- **Botnet**: a network of infected computers controlled by a remote criminal;
- **Rootkit**: malicious application attackers use to conceal their activities and maintain control over a host.

Organizations like NIST [20] and ENISA [21] recognize these malware types. In literature, three common properties describe malware: (i) propagation method, categorizing based on spread and purpose; (ii) concealment strategy, focusing on hiding tactics against users and detection; and (iii) data structure manipulation, dealing with software vulnerability exploitation. Table 6 organizes malware based on these categories.

Regarding concealment strategy, malware can be categorized into two main groups: (i) no concealment and (ii) stealthy malware [22], [23], [24]. No concealed malicious code lacks techniques to hide itself, making it easy to detect.

**TABLE 1. Malware categories based on propagation method, concealment strategy, and data structure manipulation.**

| Malware Type | Propagation Method | Concealment Strategy | Data Structure Manipulation |
|---|---|---|---|
| Worms | Network-based transmission | Polymorphisms or metamorphism | Exploitation of memory corruption vulnerabilities |
| Viruses | File-based transmission | Polymorphism | Manipulation of data structures |
| Trojans | Social engineering | No concealment | - |
| Spyware | Internet downloads | Encryption | - |
| Ransomware | Email attachments | Encryption | File system manipulation |
| Adware | Software bundling | No concealment | - |
| Rootkits | Kernel-level exploits | Obfuscation | Manipulation of system structures |
| Backdoors | Remote access | Encryption | - |
| Keyloggers | Phishing, infected software | Encryption | - |
| Botnets | Exploitation, social engineering | Encryption and polymorphism | - |

However, as shown in Table 1, only a small subset of malware does not employ concealment. File infectors like traditional viruses or worms may not heavily focus on concealment, spreading by attaching to executable files. Adware may not invest heavily in hiding and may rely on user interactions. Similarly, if achieved without sophisticated evasion, simple trojans may prioritize their primary goal over concealment.

Conversely, stealthy malware is a general term for all kinds of malicious code capable of hiding from users and detection mechanisms [25], [26]. Its primary purpose is to remain undetected for an extended period in the computing system, allowing compromising computers and stealing information before a suitable detection mechanism can be deployed to protect against it. In general, the concealment actions aim to hide the malware's trails or code. Stealthy malware may employ several techniques:

- **Encryption/obfuscation**: the oldest and simplest technique consists of a decryptor and an encrypted main body. When the infected file runs, the decryptor recovers the main body. The malware may use a different key for each infection to hide its signature, making the encrypted part unique. The decryptor small size compared to the main body reduces detection probability. Encryption complexity ranges from basic operations to strong encryption methods [22], [23], [27];
- **Oligomorphism and polymorphism**: the encryption technique limitation lies in the constant decryptor across exploitations, enabling detection based on code patterns. Oligomorphism employs a small set of decryptors, using a different one for each infection. Polymorphism, similar but with theoretically infinite decryptor variations, relies on obfuscation methods like dead-code insertion and register reassignment for distinct decryptor creation [22], [23], [28], [29];
- **Metamorphism**: the binary sequence is altered by making a new malware version for each new infection through a mutation engine. The mutation engine uses code transforming and obfuscation to change the malicious code [22], [23], [30].

Several classes of software vulnerabilities can be explored to perform security attacks. This paper focuses on the prevalent memory errors enabling memory corruption for security attacks [31], which lead to two main exploit categories: control-flow attacks and data-only attacks.

Control-flow attacks are common, easy to construct, and demand minimal application-specific knowledge. They exploit vulnerabilities like buffer overflows or injection attacks to redirect the program's execution flow, enabling arbitrary code execution [15], [32], [33], [34], [35]. Techniques such as code injection [36], Return-Oriented Programming (ROP) [37], or Jump-Oriented Programming (JOP) [38] divert execution to specific memory locations housing malicious code, bypassing standard security measures.

In contrast, data-only attacks are rarer, subtler, and require advanced knowledge of program semantics. They manipulate critical data while maintaining a valid control flow, compromising target programs without injecting additional code. These attacks alter essential data elements, such as identification or configuration data, influencing target application behaviors during runtime [39].

## III. OVERVIEW OF MALWARE DETECTION

Malware detection involves determining whether a given program exhibits malicious intent. Figure 1 offers an overview of contemporary solutions for malware detection, categorized into two main groups: software-based and hardware-based approaches. This division is rooted in differing observation points within the system stack and different detection methodologies. Recent advancements, as underscored by [13] and [18], increasingly rely on ML or Artificial Intelligence (AI) techniques to facilitate detection.

This section presents an overview of software-based and hardware-based malware detection (sub-sections III-B and III-C), starting by reviewing the metrics used for evaluating the performance and efficiency of the detectors (sub-section III-A).

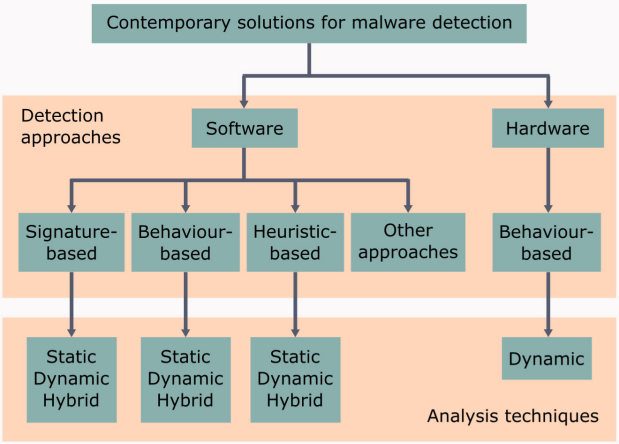


**FIGURE 1.** Overview of the contemporary solutions for malware detection. Elaborated by the authors based on [40].

### A. EVALUATION METRICS

Before delving into specific malware detection techniques, readers need to consider the evaluation metrics used to assess their effectiveness. These metrics serve as quality indicators, pivotal in determining the adoption of a technique on a commercial scale. Since malware detection is a classification problem, the quality evaluation of the detectors is based on the standard classification metrics. They can be grouped as performance metrics and efficiency metrics. Performance is the degree to which a system or component accomplishes its designated functions within given constraints, i.e., correctly detects the malware. Efficiency is the degree to which a system or component performs its specified functions with minimum consumption of resources [41].

The primary evaluation tool for performance is the confusion matrix. This matrix is fundamental in ML and classification tasks, summarizing results in a tabular form. It comprises four elements (see Table 2): True Positives (TPs) represent instances where the model correctly predicts malware presence, True Negatives (TNs) indicate correct predictions of malware absence. In contrast, False Positives (FPs) and False Negatives (FNs) denote incorrect predictions of malware presence or absence, respectively.

**TABLE 2.** Confusion matrix for malware detection.

| | Predicted Negative | Predicted Positive |
|---|---|---|
| **Actual Negative** | TNs | FPs |
| **Actual Positive** | FNs | TPs |

Such a matrix allows for the definition of more descriptive metrics, and Table 3 summarizes the most common ones [7]. The *accuracy* summarizes the overall correctness of the classification model by expressing the number of correct predictions, making it one of the most widely used metrics. In scenarios where it is crucial to avoid incorrect malware predictions, *precision* provides an accurate measure of the TPs among all positive predictions. Shifting the evaluation focus to ensure no malware passes unnoticed, the *True Positive Rate (TPR)* (also known as *Recall* or *Sensitivity*) weighs TPs against all positive samples. It has two counterparts: (i) the *False Positive Rate (FPR)*, representing the probability of a TP being missed, and (ii) the *specificity*, also known as *True Negative Rate (TNR)*, indicating the probability of an actual negative (TN) being correctly classified. Balancing Precision and Recall is often essential, and the evaluation can be accomplished using the *F1-score*, which represents their harmonic mean.

Eventually, the *Receiver Operating Characteristic (ROC) curve* offers a visual perspective to performance evaluation. It plots the TPR against the FPR on a 2D graph, enabling a visual comparison of different models and capturing multiple classification aspects by inspecting the *Area Under the Curve (AUC)*. In simple terms, the larger the AUC, the better the model. AUC is closely related to the robustness of the classifier, indicating how effectively the classifier distinguishes between malware and benign applications.

**TABLE 3.** Most common metrics for performance evaluation of classification.

| Matrix | Expression |
|---|---|
| **Accuracy (A)** | $A = \frac{TP+TN}{TP+TN+FP+FN}$ |
| **Precision (P)** | $P = \frac{TP}{TP+FP}$ |
| **TPR** | $TPR = \frac{TP}{TP+FN}$ |
| **FPR** | $FPR = \frac{FN}{FN+TP}$ |
| **Specificity (S)** | $S = \frac{TN}{TN+FP}$ |
| **F1-Score (F1)** | $F1 = \frac{2\times(P\times R)}{(P+R)}$ |
| **ROC** | $ROC = 1 - S = \frac{FP}{FP+TN}$ |
| **AUC** | $AUC = \int_0^1 R(FPR)\, dFPR$ |

According to [41], efficiency is related to the resources used for malware detection. Many metrics can be used to evaluate the efficiency [42], but in the malware detection field, latency, power consumption, and hardware cost are the main interest:

- **Latency** is the time between collecting all features analyzed by the malware detector and concluding its detection. A low latency is vital for run-time detection of malware that acts in a short interval of time;
- **Power consumption** indicates the energy the detector consumes per unit of time. Two factors primarily impact the power consumption of the detector: the hardware that implements or where the classifier runs and the detection algorithm (those with higher computing processing tend to consume more);
- **Hardware cost** indicates the monetary cost of building the detection system. This is important from both an

industry and a research perspective to dictate whether a system is financially viable. The main parameter to evaluate the hardware cost is the chip area (usually reported in square millimeters) in conjunction with the process technology (for example, 45 nm). Sometimes, the amount of memory is also used to evaluate the hardware cost.

### B. SOFTWARE-BASED MALWARE DETECTION

Software-based protection relies on specific software running in the system and analyzing the potential malware presence using different approaches. Authors in [13] and [40] proposed a very comprehensive selection of them:

- **Signature-based**: the signature is a unique malware feature extracted from structural properties (e.g., code sequences) or run-time properties [43]. The detection works as follows: features extracted from the executable generate a signature stored in a signature database. When the system is required to classify a potential threat, the detector extracts the related features and computes the signature, comparing it with signatures on the database. The potential threat is marked as malware if a hit occurs during the comparison. This approach is widely used within commercial antivirus and does not allow zero-day detection [13];
- **Software behavior analysis**: this approach is based on dynamic characteristics from run-time executions of programs [40]. Dynamic characteristics might include processor and memory information, kernel usage (system calls), file system activities, and network communications. They are extracted with monitoring tools, a dataset is created, and a ML detector distinguishes malicious and harmless applications. Software behavior analysis can detect malware variants often missed by the signature-based approach;
- **Heuristic-based detection**: this method relies on experiences and techniques, including rules and ML. The process involves two phases: first, the detector system is trained with normal and abnormal data to identify relevant characteristics. In the second phase, known as monitoring or detection, the trained detector intelligently assesses new samples to make decisions [44];
- **Deep Learning**: this falls under the umbrella of ML algorithms, enabling computational models with multiple layers to extract more advanced features from raw input [13]. The FE aspect combines elements from previous approaches, making it a novel method. Additionally, it proves highly effective for zero-day detection, as the FE, employing multiple techniques, facilitates context adaptation and model updates, as highlighted in [13].

Regarding software-based detection, it is also crucial to distinguish among the types of analysis carried out to extract the required information. According to [43], three ways are possible: (i) via *static* analysis, using syntax or structural properties of the program/process (e.g., code sequences), (ii) via *dynamic* analysis, extracting the necessary data during or after program execution, leveraging run-time information, and (iii) via *hybrid* analysis, combining the two previous. Selecting one of those also affects the expected latency of the detection. While a static analysis aims to detect the threat even before executing the malicious program, the other two might require an entire execution before detection.

### C. HARDWARE-BASED MALWARE DETECTION

Hardware-based detection, or HMD, addresses the performance and computational overhead challenges of traditional malware detection techniques by utilizing low-level micro-architectural features of running applications on the target system [18]. The concept that malware can be identified through micro-architecture hardware events stems from the observation that programs exhibit phase behaviors [16], [17]. Program phases, which vary significantly between programs, manifest as patterns in architectural and micro-architectural events. This variation enables the discrimination of programs based on their time-behavioral hardware event patterns, facilitating the differentiation between malicious and benign applications. In 2011, Malone et al. [14] demonstrated the feasibility of detecting program code modifications based on the deviation of hardware events. In 2013, Demme et al. [15] showed the feasibility of detecting Android malware and Linux rootkits using hardware events values analyzed by a ML classifier.

The idea of HMD is to perform dynamic analysis leveraging micro-architecture hardware events monitored by most modern microprocessors using Hardware Performance Counters (HPCs) [45]. Various ML techniques can be applied to the HPCs collected data [18]. One of the primary advantages of HMD is that the analysis relies on real-time hardware collected data, enabling fast ML classification; a few milliseconds suffice to identify threats. This translates to low latency, enabling runtime detection [46], [47], [48]. Unlike static technique analysis employed by most software-based antivirus solutions, which can be easily subverted by stealthy malware using concealment techniques, dynamic analysis via hardware-based approaches facilitates the detection of code variants and unknown malware [15]. Moreover, while software-based detection tools are software-based and susceptible to bugs or oversights in the underlying system software, hardware-based detection with secure hardware significantly reduces the possibility of malware subverting protection mechanisms [15], [34].

On the performance front, the dynamic analysis conducted by software-based detection necessitates sophisticated computation, often at the expense of significant performance overhead. The increasing software size further complicates dynamic software analysis [15]. Conversely, in the hardware-based approach, understanding software behavior provided by micro-architectural events simplifies the analysis, reducing computational processing efforts and the cost of hardware-based detection [15], [49].

However, while the HPCs demonstrate their ability to track behavioral deviations [50], [51], [52], their effectiveness remains open to discussion. On the positive side, [15], [34] demonstrated detector performance using this approach, reporting accuracy consistently exceeding 80%, deeming it effective. Conversely, [53] and [54] conducted experiments challenging the effectiveness of hardware-based detection. They argued that reported detection capabilities often stem from tiny sample sizes and experimental setups favoring the detection mechanism unrealistically. Even if accurate, an 80% accuracy is insufficient in scenarios with thousands of executables, risking many benign applications being misclassified as malware. They also questioned the causal link between low-level micro-architectural events and high-level software behavior. Lastly, they illustrated the hardware-based detector inability to distinguish ransomware embedded in a benign application like Notepad++. In a recent contribution, [55] acknowledged the absence of a perfect malware detector and argued that hardware-based detection is only effective for specific malware types. In particular, [55] proposes its effectiveness in identifying attacks exploiting architectural side-effects, citing examples such as RowHammer [56], [57] (detectable through excessive cache flushes [58]), ROP attacks [37] (identified by an abundance of instruction misses [59]), and DirtyCoW [60] (detectable through heightened paging activity). The authors also emphasized the necessity for a maliciousness theory to enhance the understanding of malware threats and assess proposed defenses.

While HPCs have been used in the past for safety and security, performance analysis, and optimization [61], [62], [63], it is well-known that they may suffer from inconsistency in implementation, leading to non-determinism and overcounting [64]. Das et al. highlighted some of these HPC challenges in security [65]. Recent studies address HPC discrepancies, propose methodologies, analyze resilience, and compare HPCs in various machines [66], [67], [68], [69]. Given that HPCs are hardware-based protections, detectors may be designed for specific devices with characteristics defined by the architecture and manufacturer. For instance, processors may track different numbers of events simultaneously, and discrepancies in instruction counting methods are possible [61]. These factors underscore the need for malware detection applications to abstract software from the hardware level.

Among the inconsistencies and limitations of HPCs, some countermeasures can be deployed to stabilize the generated data [61], [65]. They include per-process filtering of events (applied by saving and restoring the counter values at context switches), proper interrupt handling, and minimizing the impact of non-deterministic events. In general, all works acknowledge that the evolution and improvement of the processors hardware monitoring units also tend to reduce this issue. Eventually, the classification task built on top of the HPC data is commonly a ML one. This frequently leads to techniques that increase the complexity of such algorithms, like ensemble learning and time series or even Deep Neural Networks (DNNs) [18].

## IV. HARDWARE-BASED MALWARE DETECTION BASICS

This section focuses on HMD techniques, outlining their key components.

### A. HARDWARE EVENTS AND PERFORMANCE COUNTERS

Modern processors have units to monitor hardware events. In 2002, Sprunt [70] published a seminal paper on the basics of Performance Monitoring Units (PMUs). These units were developed to collect data about the performance of applications, operating systems, and processors and to help programmers tune algorithms and codes. Software dynamically adjusted to resource utilization would also benefit from the information collected. The proven advantages of utilizing the PMUs, the continuous improvements of these units, and their constant spreading among different devices have led to their leverage for safety and security purposes [50], [52], [62], [63].

Nowadays, PMUs can monitor several hardware events (see Figure 2). Complex devices like high-end processors have hundreds of events to monitor. These events include retired instructions (branches, load, store, etc.), branch predictions, cache hits and misses, floating-point operations, hardware interrupts, elapsed core clock ticks, core frequency, and temperature. However, to minimize hardware complexity, only a few HPCs (e.g., 2 to 8 in high-end processors) are generally available, thus limiting the number of parallel events that can be monitored. Each HPC has an event detector and an associated counter [71].

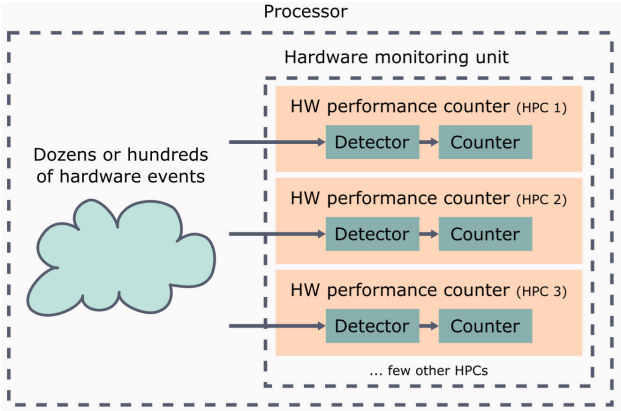


**FIGURE 2.** Hardware events and performance counters in a processor. Elaborated by the author.

### B. HARDWARE-BASED DETECTION FRAMEWORK

A generic framework can be a guiding structure to facilitate the implementation of HMD, as illustrated in Figure 3. The framework leverages the existing PMU within the processor and consists of two primary components: (i) data collection

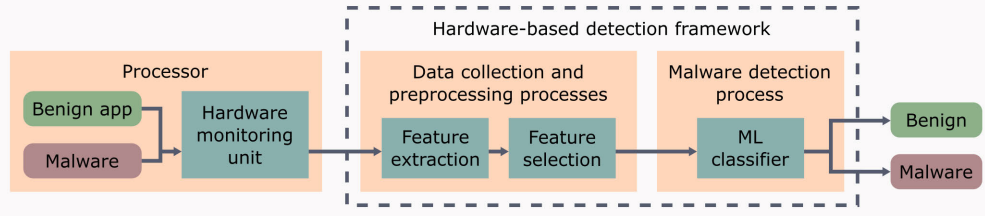


**FIGURE 3.** A generic hardware-based detection framework. Elaborated by the author.

and preprocessing and (ii) malware detection. This section provides a detailed overview of the implementation process.

Data collection involves FE and Feature Selection (FS) [72], [73]. FE captures and stores HPCs in a vector space, enabling the FS to select a subset that efficiently describes the input data while minimizing noise and irrelevant variables, ensuring optimal prediction results. FE can occur in the time or event domain [70]. In the time-based domain, the application execution is periodically interrupted to record HPC values. Conversely, the event domain triggers interruptions based on specific events or a set number of executed instructions rather than regular intervals.

In terms of strategies to perform FE, we envision four alternatives: (i) instrument the source code with the employment of a library, like `PAPI` [74]; (ii) develop of a proprietary kernel module or driver, as performed in [34]; (iii) use of an available utility that performs tasks mainly in the OS kernel, like `PERF` [75]; and (iv) use of a micro-architectural simulator to model the processor as it executes the application, like `gem5` [76] and `GVSoC` [77].

During FE, the sampling strategy is crucial. In the time-based domain, parameters such as period, frequency, or number of cycles determine when HPCs are sampled. In the event-based domain, sampling depends on the number of event or instruction occurrences. The chosen FE strategy influences these definitions. A proprietary kernel module or driver allows programmers to choose between time-based or event-based domains, set parameters for sampling triggering, and specify values. However, configurations are limited when libraries like `PAPI` and `PERF` are used. Regarding sampling values, in time-based sampling, there is no fixed ideal period or frequency, varying based on the experiment and goal. Hardware-based detection experiments typically use periods in the order of milliseconds or seconds. Striking a balance between low and high sampling frequencies is essential, considering the trade-off between computational processing, data quantity, and system effects.

FS offers multiple advantages, including addressing the Curse of Dimensionality in ML [78], enhancing data understanding, reducing computation requirements, and improving predictor performance. Filter-based algorithms dominate the FS in the HMD field, ranking features based on a scoring criterion, using a threshold for variable selection. They are valued for simplicity and practical application success, focusing on the relevancy of features. Prominent methods include Principal Component Analysis (PCA) (used by [47], [53], [79], and [80]), Fisher Score [81] (used by [34] and [51]), Pearson Correlation Coefficient [82] (used by [46], [47], [48], [79], and [80]) and Information Gain (Mutual Information) [83] (used by [84] and [85]). The Scikit-learn [86] library for the Python and Weka [87] are tools frequently used in the HMD field for FS.

Since the number of events that can be potentially monitored exceeds the available HPCs, some studies (for example, [14], [32], [48]) also perform a preliminary manual FS before data collection, thus reducing the number of software executions required to collect data. The selection is based on architectural and micro-architectural knowledge and other studies.

Eventually, in HMD, ML algorithms play a crucial role. Supervised and unsupervised learning techniques are employed in hardware-based malware detection. While for supervised detection, both benign and malignant samples, adequately annotated, are necessary, in unsupervised malware detection, the classifier is trained only with benign applications to perform anomaly detection [88]. Unsupervised detection has two exciting advantages: (i) it does not require a malware dataset for training, and (ii) the classifier can detect zero-day malware [18]. On the other side, unsupervised algorithms are complex, requiring more sophisticated analysis and resulting in complex hardware implementations.

Several traditional classification algorithm families are employed in HMD: linear regression (LinearRegression and SimpleLinearRegression), logistic regression (Logistic and SimpleLogistic), Bayesian network (BayesNet and NaiveBayes), decision trees (J48 and REPTree), rule-based (JRIP, OneR and PART), Artificial Neural Network (ANN) (MultiLayerPerceptron), K-Nearest Neighbors (KNN) (IBk), ensemble learning (AdaBoostM1, Bagging and RandomForest) and Support Vector Machines (SVM) (SMO) [78]. The algorithms in parentheses refer to specific Weka implementations, which are commonly used in the context of HMD. Further details on these families and their implementations in Weka can be found in [87].

Eventually, a crucial consideration is the trade-off between monitoring more events for better application characterization and detector performance and the impact on runtime applicability. Some studies used many events, exceeding available HPCs, necessitating multiple application runs [15],

[84], [89]. This trade-off is further addressed in ML solutions discussed in Section V-C.

## V. HARDWARE-BASED DETECTION ASSESSMENT

The following sections analyze the performance and efficiency of the state-of-the-art in HMD and explore ML techniques to enhance detector performance.

### A. PERFORMANCE

Tables 4 and 5 provide a comprehensive overview of the literature contributions in the field, aiming to facilitate fair comparisons by presenting the best-case results in Table 4. Metrics were directly sourced from the paper's text whenever feasible, with manual extraction from reported ROC curves employed only when necessary. The "Classification" column denotes the classification algorithm associated with the best result, with the Weka implementation serving as a reference. Conversely, Table 5 outlines, for each contribution in Table 4, the range of considered scenarios in terms of malware, classifiers, and system characteristics. The values in Table 4 underscore the efficacy of HMD in supporting malware detection and highlight the overall high quality of the findings.

Among all contributions reported in Table 4, authors in [93] showcase the effectiveness of HMD on real scenarios: DARPA Rapid Attack Detection, Isolation and Characterization Systems (RADICS), Intel Threat Detection Technology (TDT), and Microsoft Defender. This is a tangible exploitation of HMD into actual products. Still, using a single type of classifier (i.e., SVM) leaves room for research and improvements.

As most of the current works on HMD rely on ML classifiers, the analysis conducted by Patel et al. [48], summarized in Table 6, is particularly interesting. The authors thoroughly analyze eleven ML classification algorithms (based on Weka [87] implementations). The goal was to understand the trade-offs between the design parameters offered by the algorithms. The chosen metric to evaluate performance was accuracy. The dataset used for training and testing the algorithms was extracted using the PERF tool in intervals of 10 ms executed in an Intel Haswell Core i5-4590 processor running Ubuntu 14.04 with Linux kernel 4.4. The baseline of benign application comprises the Mibench benchmark suite [94], Linux system programs, browsers, text editors, and word processors. The malware came from the VirusTotal dataset. Since the HPCs available in an Intel architecture are considerable, the accuracy of ML algorithms covers different numbers (i.e., 32, 8, 4, 2, and 1) of hardware events. Table 6 reports the accuracy for 4 hardware events, a reasonable quantity for concurrent monitoring in most modern processors, even in embedded scenarios [50]. JRIP (rule-based) presented the top accuracy, followed by four classifiers with the same top-two accuracy: J48 (decision-tree), OneR and PART (rule-based), and SGD. In this case, most classifiers have accuracy above 80%. Another interesting observation is that reducing the hardware events below four significantly impacts the performance of most classifiers.

Similar findings are reported in Torres and Liu [51]. While the authors concentrated on a particular malware subclass (data-only exploits from [92]), they implemented two different experiments on different classifiers, distinguishing between using the complete set of 50 features or a smaller set of 6 features. The findings report a very high accuracy on the complete set of features (as seen on the first of the two rows dedicated to the paper in Table 4) and a degradation when only a subset is used.

### B. EFFICIENCY

Alongside the detection quality, the HMD aims to reduce the detectors cost in terms of resources. As the data required for the classification come from the hardware layer of the system stack, most studies evaluate FPGA-based implementations of ML classifiers, providing measures for the power consumption and the area as the goal is to understand the trade-offs between the design parameters offered by the algorithms. When the classifier is software-based, the evaluation usually includes the latency, avoiding further monitoring of other resources. Unfortunately, as seen in Table 4, not all works report the latency of the detection or, more in general, the costs of it. Generally, whenever the detection is performed at the software level, the latency is less than 1 ms. At the same time, more optimized hardware implementations can scale down to tens or hundreds of ns.

As reported in the previous section, the work from Patel et al. [48] covered a thorough analysis and, for this reason, is undoubtedly an excellent candidate to show the efficiency of the methodology. For hardware implementation, authors used the Xilinx Virtex 7 FPGA, implemented Weka models in C code, and used the Xilinx High-Level Synthesis (HLS) compiler to generate the final bitstream. The latency was evaluated both in software and hardware implementations. Authors implemented the classification algorithms in software at the OS kernel level, which includes the time to read the HPC and execute the classifiers. Eventually, the Intel Turbo Boost technology was disabled, as it might introduce errors in the time measurement, and the CPU governor was operating at a constant frequency of 800 MHz. The IP cores with the algorithms were synthesized in Vivado to estimate the power consumption, considering a 100 MHz clock. Power estimation contains both static power and dynamic power consumption of digital logic.

Values in Table 6 show the considerable difference between the latencies in software and hardware implementations. Software implementations have latencies almost in the order of milliseconds (ranging from 0.624ms to 0.870ms, best and worst cases). In contrast, hardware implementations are in the order of nanoseconds (ranging, in this case, from 10ns to 3020ns). The authors underlined that these slow profiles displayed by classifiers in the kernel space are three orders bigger than several malware executions (ranging in microseconds). Other findings related to latency are crucial

**TABLE 4.** Summary of best-case performance from main studies in the hardware-based malware detection approach. # HPCs column refers to the number of hardware events the classifiers consider. Classification algorithm labels are based on Weka implementations used in the referenced studies. Evaluation metrics as defined in Section III-A: A is Accuracy, P is Precision, S is Specificity, and F1 is the F1-Score.

| Year | Ref. | Target | # HPCs | Classification | Learning | Latency | Evaluation Metrics | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | A | P | TPR | S | F1 | AUC |
| 2013 | [15] | Android malware | 6 | Decision Tree | Offline | NA | - | - | - | - | - | 0.83 |
| | | Linux rootkits | 4 | KNN | Offline | NA | - | - | 0.70[1] | - | - | - |
| 2014 | [34] | Internet Explorer exploitation | 4 | SVM | Offline | NA | - | - | - | - | - | 1.00 |
| | | Adobe PDF Reader exploitation | 4 | SVM | Offline | NA | - | - | - | - | - | 1.00 |
| 2015 | [32] | Ransomware | 5 | Logistic regression | Offline | NA | - | - | - | - | - | 0.94 |
| | | Ransomware | 5 | Logistic regression (with Specialization) | Offline | NA | 0.87 | - | 0.81 | 0.96 | - | - |
| 2015 | [33] | Viruses, worms, trojan horses, spyware, adware, and botnets | 5 | ANN | Offline | NA | - | - | 1.00[1] | - | - | - |
| 2017 | [48] | Malware from various categories, sourced from VirusTotal [90] dataset | 4 | BayesNet | Offline | 0.624ms (SW) / 140ns (HW) | 0.85 | - | - | - | - | - |
| 2017 | [84] | Rootkits | 16 | SVM | | NA | 1.00 | 1.00 | 1.00 | - | 1.00 | - |
| 2018 | [46] | Malware from various categories, sourced from VirusTotal [90] dataset | 4 | J48 (with Ensemble Learning) | Offline | NA | 0.83 | - | - | - | - | 0.94 |
| 2018 | [53] | Malware from various categories, sourced from VirusTotal [90] dataset | 6 | Random Forest | Offline | NA | - | 0.86 | 0.83 | - | 0.85 | 0.92 |
| 2019 | [65] | Malware from various categories, sourced from VX Heaven [91] dataset | 5 | J48 | Offline | NA | - | 0.82 | 0.82 | - | 0.82 | 0.93 |
| 2019 | [47] | Backdoor | 4 | OneR | Offline | NA | - | - | - | - | 0.94 | - |
| | | Rookit | 4 | MLP | Offline | NA | - | - | - | - | 0.94 | - |
| | | Virus | 4 | J48 and ensemble learning (AdaBoostM1) | Offline | NA | - | - | - | - | 0.96 | - |
| | | Trojan | 4 | MLP | Offline | NA | - | - | - | - | 0.99 | - |
| 2021 | [79] | Trojan | 4 | JRIP | Offline | 20ns | - | 0.93 | - | - | - | - |
| 2021 | [80] | Stealthy rootkits | 4 | DNN | Offline | NA | 0.93 | 0.95 | 0.90 | - | 0.93 | 0.98 |
| 2022 | [51] | Data-only exploits [92] | 50[2] | Two Classes-SVM | Offline | NA | 0.99 | - | - | - | - | - |
| | | Data-only exploits [92] | 6 | LZ78 | Offline | NA | 0.84 | - | - | - | - | - |
| 2022 | [93] | Stealthy attack on power grid | 6 | SVM | Offline | 120s | 0.94 | - | - | - | - | - |

[1] Values extracted from ROC curves considering a false positive rate of 10%.
[2] 50 is the whole set of features. This is why the authors also investigated a reduced set (in the following line).

to highlight. In software implementations, the latency for reading the HPC is negligible when monitoring a single core but may increase significantly when monitoring multiple cores. Moreover, the more HPCs to read, the longer it takes. Concerning the classification algorithms, BayesNet (Bayesian network), PART (rule-based), and SimpleLogistic (logistic regression) showed the lowest latency values when implemented in software. Conversely, none of these three are on the list of the top three low latencies in hardware. NaiveBayes (Bayesian network), MLP (ANN), and J48 (decision tree) are the three best hardware implementations. This paradox demonstrates the uncorrelation between the algorithms' latencies when comparing implementations at the kernel space and hardware.

### C. MACHINE LEARNING TECHNIQUES CONSIDERATIONS

Recent studies have explored various ML methods to enhance the performance of HMD detection approaches, especially in the last five years. These techniques aim to overcome the challenge of limited application characterization due to the concurrent capacity of PMUs to monitor hardware events. While these methods show performance improvements, they

**TABLE 5.** Reference studies including details on the full list of targets and classifications approaches tested and details on the reference systems.

| Year | Ref. | Targets | Classification | Devices | OS |
|---|---|---|---|---|---|
| 2013 | [15] | Android malware, Linux rootkits | Decision trees, ANN, KNN, Random Forest | Arm Cortex-A9 OMAP4460, Intel Xeon X5550 | Android 4.1.1-1 (kernel 3.2), Linux kernel 2.6.32 |
| 2014 | [34] | Exploitations on Internet Explorer and Adobe PDF Reader | SVM | Intel IvyBridge Core i7 | Windows XP |
| 2015 | [32] | Ransomware, password stealers, trojan horses, backdoor, worms | Logistic regression (w/o specialization) | Not specified | Windows 7 |
| 2015 | [33] | Viruses, worms, trojan horses, spyware, adware, and botnets | ANN | Not specified, Altera EP4CE115 | Windows 7 |
| 2017 | [48] | Malware from various categories, sourced from VirusTotal [90] dataset | Logistic, SimpleLogistic, BayesNet, NaiveBayes, J48, PART, JRIP, OneR, MultiLayerPerceptron, SMO, SGD | Intel Haswell Core i5-4590, Xilinx Virtex 7 | Ubuntu 14.04 (kernel 4.4) |
| 2017 | [84] | Rootkits | SVM, Decision tree, OC-SVM, Naive Bayes | Intel IvyBridge and Broadwell | Windows 7 |
| 2018 | [46] | Malware from various categories, sourced from VirusTotal [90] dataset | BayesNet, J48, REPTree, JRIP, OneR, MultiLayerPerceptron, SMO, SGD (w/o ensemble learning based on AdaBoostM1, Bagging) | Intel Xeon X5550, Xilinx Virtex 7 | Ubuntu 14.04 (kernel 4.4) |
| 2018 | [53] | Malware from various categories, sourced from VirusTotal [90] dataset | Decision trees, Naive Bayes, ANN, KNN, Random Forest (w/o ensemble learning AdaBoost) | AMD Bulldozer | Windows 7 |
| 2019 | [65] | Malware from various categories, sourced from VX Heaven [91] dataset | J48, IBk, SMO | Intel Sandy Bridge, Haswell, and Skylake | Ubuntu 16.04 |
| 2019 | [47] | Backdoor, rootkits, viruses, trojan horses | J48, JRIP, OneR, MLP (w/o AdaBoostM1) | Intel Xeon X5550, Xilinx Virtex 7 | Ubuntu 14.04 (kernel 4.4) |
| 2021 | [79] | Worms, rootkits, viruses, trojan horses | REPTree, JRIP, OneR, MLP, SGD | Intel Xeon X5550, Xilinx Virtex 7 | Ubuntu 14.04 (kernel 4.4) |
| 2021 | [80] | Stealthy backdoor, rootkits, and trojan horses | DNN | Intel Xeon X5550 | Ubuntu 14.04 (kernel 4.4) |
| 2022 | [51] | Data-only exploitation | TC-SVM, OC-SVM, LZ78 | Intel Nehalem Core i7-920 | Ubuntu 16.04 (kernel 4.13) |
| 2022 | [93] | Stealthy attack on power grid | SVM | OpenPLC controller with Raspberry PI | 8-bus power grid in a PowerWorld simulator |

often introduce increased complexity in classifiers, resulting in reduced efficiency, i.e., higher power consumption and increased area requirements. This section discusses ensemble learning, specialization, adaptive detection, and time series ML techniques in HMD.

In ensemble learning, multiple ML algorithms are trained separately to create a classifier, combining their results to improve decision accuracy [95]. In HMD, ensemble classifiers leverage the characteristics of individual algorithms to detect various types of malware while minimizing hardware events for runtime detection [32], [46], [47]. However, the performance gains come with increased complexity and efficiency overhead [46], [79].

Sayadi et al. [46] assessed the efficiency impact of ensemble learning in a malware detector on Xilinx Virtex 7 FPGA. Significant latency increases were observed when comparing a general classifier with 8 HPCs to a Boosted classifier [96] with 4 HPCs. When Boosted, the general MLP algorithm passed from a latency of 3020ns to a latency of 5910ns. OneR increased from 10ns to 700ns, and J48 increased from 90ns to 670ns. In terms of hardware cost, the largest area increases were observed in OneR (from 2.1% to 5.1%), JRIP (from 2.5% to 5.3%), and BayesNet (from 11.5% to 13.6%). Conversely, J48, REPTree, and MLP showed smaller area increases. The findings highlight substantial overhead in both latency and hardware costs.

Another interesting ML technique is the specialization. Instead of training a single multi-class classifier able to recognize several malware categories, different classifiers are trained, each specialized in detecting a specific malware. Authors in [32] discuss and explore specialized detectors in HMD. They used a logistic regression-based classifier

**TABLE 6. Performance and efficiency of classifiers based on Weka implementations. Extracted from [48].**

| Classifier | Accuracy | SW Latency (ms) | Latency (ns) | HW Power (W) | Area[1] |
|---|---|---|---|---|---|
| BayesNet | 81.13 | 0.624 | 140 | 0.44 | 6794 |
| J48 | 82.07 | 0.663 | 60 | 0.44 | 1801 |
| JRIP | 83.96 | 0.653 | 90 | 0.44 | 1504 |
| Logistic | 79.24 | 0.844 | 340 | 0.63 | 13041 |
| MLP | 81.13 | 0.870 | 40 | 1.03 | 36252 |
| NaiveBayes | 78.30 | 0.802 | 10 | 1.34 | 58177 |
| OneR | 82.07 | 0.653 | 220 | 0.32 | 1258 |
| PART | 82.07 | 0.642 | 680 | 0.44 | 2131 |
| SGD | 82.07 | 0.652 | 340 | 0.44 | 2556 |
| SimpleLogistic | 79.24 | 0.648 | 3020 | 0.45 | 4721 |
| SMO | 73.58 | 0.652 | 2330 | 0.44 | 2556 |

[1] The area is a function of total lookup tables, flip-flops, and DSP blocks.

for each malware class. As a result, the proposed detectors reduced the false positive rate by more than half compared to a single detector while increasing the detection rate. The authors proposed a two-level detector in the same paper, mixing a first level based on the hardware detection approach and a second level based on the software detection approach. The hardware detector was based on specialized ensemble techniques. The latency of this scheme was compared with malware detection purely based on software methods. As a result, they reported average latency reduced to 1/6.6 when the fraction of malware is low and latency reduced to 1/3.1 when 20% of the programs are malware.

In 2019, Sayadi et al. [47] introduced a specialized two-stage malware detector, leveraging ensemble learning techniques, significantly improving accuracy. The first stage classifies applications into benign or malware categories (virus, rootkit, backdoor, and trojan horse). The second stage deploys an ML classification algorithm that works best for each category of malware. Their 2021 work [80] continued using specialization for an accurate and run-time stealthy malware detector. They also evaluated the efficiency overhead of their specialized and ensemble learning malware detector, implemented on Xilinx Virtex 7 FPGA. A comparison of a general classifier with 4 HPCs to a Boosted classifier with 4 HPCs revealed notable latency increases for MLP (from 1.020 to 5.910 ms), OneR (from 10 to 700 ns), J48 (from 30 to 670 ns), and JRIP (from 20 to 560 ns). MLP (from 43.2% to 61.7%), JRIP (from 0.26% to 5.3%), OneR (from 0.49% to 5.1%) and J48 (from 0.93% to 4.3%) exhibited considerable increases regarding hardware cost. The findings emphasize substantial latency and hardware cost overhead.

Adaptive detection was proposed by Gao et al. [79] to optimize the performance versus cost. It targets higher or similar performance as ensemble learning, with a reduced cost. The technique leverages the concept that the ML algorithm employed in the detector strongly correlates both the nature of the scrutinized malware and the overall performance metric. Adaptive detection involves a dynamic framework that assesses all underlying ML algorithms in real time, opting for the optimal classifier to identify malicious patterns effectively. The implementation encompasses two primary online stages: (i) algorithm selection and (ii) malware detection. Consequently, only the most efficient ML-based detector is employed to differentiate malware from the benign class, eliminating the need to acquire results from individual base detectors and enhancing overall efficiency.

In the adaptive detector proposed by Gao et al. [79], the algorithm selection step is done by a lightweight tree-based decision-making algorithm that accurately selects the most efficient model for inference. As a result, the scheme showed up to a 94% detection rate while improving the cost-efficiency by more than 5X compared to existing ensemble-based malware detection methods.

Eventually, time series classification is fundamental to understanding the key concept behind hardware-based malware detection. The intuition driving this technique stems from the program's phase behavior, transforming malware detection into a time series classification problem. In addressing this challenge, Sayadi et al., as outlined in [97] and [80], introduced a time series ML technique designed to identify stealthy malware in real time. In scenarios where attackers embed malicious files within benign programs on target hosts, executing both applications as a single thread, traditional signature-based antivirus tools falter. Embedded malware remains elusive even when the exact malware signature is in the detector database. The authors proposed a classifier based on a Fully Convolutional Neural Networks (FCNNs) and exclusively utilized branch instructions as a low-level feature in their solution. The results demonstrated the efficacy of their technique, achieving a remarkable average detection performance of 94% with only one HPC feature, surpassing state-of-the-art detection methods. This enhanced performance, however, comes at a higher computational cost associated with employing a deep-learning-based solution.

While not explicitly implementing a time series technique, also [93] reports similar results on the Intel TDT use case. Although no specific numbers are provided, the paper compares the Fast Fourier Transform (FFT) counting traces of the branch instructions and branch misprediction events for the WannaCry ransomware, underlining the significant difference with or without the ransomware.

## VI. CONCLUSION AND RESEARCH CHALLENGES

In summary, this paper provided a comprehensive overview of HMD field, with a detailed analysis of hardware-based detection, harnessing the power of HPCs and ML. The advantages of HMD include resilience to malware subverting the protection mechanism, adaptability to code variants and

unknown malware, low complexity and overhead, potential for run-time detection, and cost reduction.

However, challenges persist in HMD. The detection accuracy is the most significant challenge as classifiers have a statistical nature. Thus, their results are not deterministic, and ongoing research aims to minimize errors by exploring complex classifiers. In cases where high accuracy is unattainable, a potential solution combines software and hardware-based detectors concurrently, with hardware as the primary defense. Moreover, ensuring consistency, accuracy, and standardization of hardware monitoring units (including HPCs) is crucial for trustworthiness. Chip manufacturers can contribute by designing appropriate modules and providing comprehensive documentation. The limited number of HPCs in mobile and IoT devices poses a feasibility challenge for this approach in these domains. Addressing these challenges will contribute to the continued advancement and effectiveness of HMD.

## REFERENCES

[1] G. McGraw and G. Morrisett, "Attacking malicious code: A report to the infosec research council," *IEEE Softw.*, vol. 17, no. 5, pp. 33–41, Sep. 2000.

[2] T. Alsmadi and N. Alqudah, "A survey on malware detection techniques," in *Proc. Int. Conf. Inf. Technol. (ICIT)*, Jul. 2021, pp. 371–376.

[3] W. Stallings and L. Brown, *Computer Security: Principles and Practice*, 3rd ed. Upper Saddle River, NJ, USA: Prentice-Hall, 2014.

[4] SonicWall. *New Sonicwall Research Finds Aggressive Growth in Ransomware, Rise in IoT Attacks*. Accessed: Sep. 18, 2023. [Online]. Available: https://www.sonicwall.com/news/new-sonicwall-research-finds-aggressive-growth-in-ransomware-rise-in-iot-attacks/

[5] (Jan. 2023). *Global Risk Report 2023*. [Online]. Available: https://www.weforum.org/publications/global-risks-report-2023/in-full/

[6] Cisco and C Ventures. *2023 Cybersecurity Almanac: 100 Facts, Figures, Predictions and Statistics*. Accessed: Sep. 18, 2023. [Online]. Available: http://cybersecurityventures.com/cybersecurity-almanac-2023/

[7] Y. Ye, T. Li, D. Adjeroh, and S. S. Iyengar, "A survey on malware detection using data mining techniques," *ACM Comput. Surv.*, vol. 50, no. 3, pp. 1–40, Jun. 2017, doi: 10.1145/3073559.

[8] D. Ucci, L. Aniello, and R. Baldoni, "Survey of machine learning techniques for malware analysis," *Comput. Secur.*, vol. 81, pp. 123–147, Mar. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404818303808

[9] 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. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804519303868

[10] J. Qiu, J. Zhang, W. Luo, L. Pan, S. Nepal, and Y. Xiang, "A survey of Android malware detection with deep neural models," *ACM Comput. Surv.*, vol. 53, no. 6, pp. 1–36, Dec. 2020, doi: 10.1145/3417978.

[11] Y. Liu, C. Tantithamthavorn, L. Li, and Y. Liu, "Deep learning for Android malware defenses: A systematic literature review," *ACM Comput. Surv.*, vol. 55, no. 8, pp. 1–36, Dec. 2022, doi: 10.1145/3544968.

[12] C. Catal, G. Giray, and B. Tekinerdogan, "Applications of deep learning for mobile malware detection: A systematic literature review," *Neural Comput. Appl.*, vol. 34, no. 2, pp. 1007–1032, Jan. 2022, doi: 10.1007/s00521-021-06597-0.

[13] F. Deldar and M. Abadi, "Deep learning for zero-day malware detection and classification: A survey," *ACM Comput. Surv.*, vol. 56, no. 2, pp. 1–37, Sep. 2023, doi: 10.1145/3605775.

[14] 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.* New York, NY, USA: Association for Computing Machinery, Oct. 2011, pp. 71–76, doi: 10.1145/2046582.2046596.

[15] 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, doi: 10.1145/2508148.2485970.

[16] T. Sherwood, E. Perelman, G. Hamerly, S. Sair, and B. Calder, "Discovering and exploiting program phases," *IEEE Micro*, vol. 23, no. 6, pp. 84–93, Nov. 2003.

[17] C. Isci, G. Contreras, and M. Martonosi, "Live, runtime phase monitoring and prediction on real systems with application to dynamic power management," in *Proc. 39th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Dec. 2006, pp. 359–370.

[18] Z. He, T. Miari, H. M. Makrani, M. Aliasgari, H. Homayoun, and H. Sayadi, "When machine learning meets hardware cybersecurity: Delving into accurate zero-day malware detection," in *Proc. 22nd Int. Symp. Quality Electron. Design (ISQED)*, Apr. 2021, pp. 85–90.

[19] M. Christodorescu, S. Jha, D. Maughan, D. Song, and C. Wang, *Advances in Information Security: Malware Detection*, 1st ed. New York, NY, USA: Springer, 2007.

[20] NIST. *Glossary*. Accessed: Mar. 7, 2023. [Online]. Available: https://csrc.nist.gov/glossary

[21] ENISA. *Botnets*. Accessed: Mar. 7, 2023. [Online]. Available: https://www.enisa.europa.eu/topics/incident-response/glossary/botnets

[22] J. Aycock, *Computer Viruses and Malware* (Advances in Information Security). New York, NY, USA: Springer, 2006.

[23] I. You and K. Yim, "Malware obfuscation techniques: A brief survey," in *Proc. Int. Conf. Broadband, Wireless Comput., Commun. Appl.*, Nov. 2010, pp. 297–300.

[24] B. Bashari Rad, M. Masrom, and S. Ibrahim, "Camouflage in malware: From encryption to metamorphism," *Int. J. Comput. Sci. Netw. Secur.*, vol. 12, pp. 74–83, Jan. 2012.

[25] S. J. Stolfo, K. Wang, and W.-J. Li, "Towards stealthy malware detection," in *Malware Detection*, M. Christodorescu, S. Jha, D. Maughan, D. Song, and C. Wang, Eds. Boston, MA, USA: Springer, 2007, pp. 231–249.

[26] E. M. Rudd, A. Rozsa, M. Günther, and T. E. Boult, "A survey of stealth malware attacks, mitigation measures, and steps toward autonomous open world solutions," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 2, pp. 1145–1172, 2nd Quart., 2017.

[27] M. Nadim, D. Akopian, and W. Lee, "A review on learning-based detection approaches of the kernel-level rootkit," in *Proc. Int. Conf. Eng. Emerg. Technol. (ICEET)*, Oct. 2021, pp. 1–6.

[28] W. Wong and M. Stamp, "Hunting for metamorphic engines," *J. Comput. Virol.*, vol. 2, no. 3, pp. 211–229, Nov. 2006.

[29] E. Konstantinou, "Metamorphic virus: Analysis and detection," Ph.D. dissertation, Dept. Math., Univ. London, London, U.K., 2008.

[30] K. Brezinski and K. Ferens, "Metamorphic malware and obfuscation: A survey of techniques, variants, and generation kits," *Secur. Commun. Netw.*, vol. 2023, pp. 1–41, Sep. 2023, doi: 10.1155/2023/8227751.

[31] V. Van der Veen, N. Dutt-Sharma, L. Cavallaro, and H. Bos, "Memory errors: The past, the present, and the future," in *Research in Attacks, Intrusions, and Defenses*, D. Balzarotti, S. J. Stolfo, and M. Cova, Eds. Berlin, Germany: Springer, 2012, pp. 86–106.

[32] K. N. Khasawneh, M. Ozsoy, C. Donovick, N. Abu-Ghazaleh, and D. Ponomarev, "Ensemble learning for low-level hardware-supported malware detection," in *Proc. 18th Int. Symp. Recent Adv. Intrusion Detect.*, vol. 9404. Berlin, Germany: Springer-Verlag, 2015, pp. 3–25, doi: 10.1007/978-3-319-26362-5.

[33] 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.

[34] 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, Eds. Cham, Switzerland: Springer, 2014, pp. 109–129.

[35] X. Wang and R. Karri, "NumChecker: Detecting kernel control-flow modifying rootkits by using hardware performance counters," in *Proc. 50th ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, May 2013, pp. 1–7.

[36] D. Ray and J. Ligatti, "Defining code-injection attacks," *ACM SIGPLAN Notices*, vol. 47, no. 1, pp. 179–190, Jan. 2012, doi: 10.1145/2103621.2103678.

[37] M. Prandini and M. Ramilli, "Return-oriented programming," *IEEE Secur. Privacy*, vol. 10, no. 6, pp. 84–87, Nov. 2012.

[38] T. Bletsch, X. Jiang, V. W. Freeh, and Z. Liang, "Jump-oriented programming: A new class of code-reuse attack," in *Proc. 6th ACM Symp. Inf., Comput. Commun. Secur.* New York, NY, USA: Association for Computing Machinery, Mar. 2011, pp. 30–40, doi: 10.1145/1966913.1966919.

[39] S. Chen, J. Xu, E. C. Sezer, P. Gauriar, and R. K. Iyer, "Non-control-data attacks are realistic threats," in *Proc. 14th Conf. USENIX Secur. Symp.*, vol. 14, 2005, p. 12.

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

[41] *ISO/IEC/IEEE International Standard—Systems and Software Engineering-vocabulary*, Standard 541, 2017.

[42] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, *Efficient Processing of Deep Neural Networks* (Synthesis Lectures on Computer Architecture Series). Cham, Switzerland: Springer, 2020.

[43] N. Idika and P. Mathur, "A survey of malware detection techniques," Purdue Univ., West Lafayette, IN, USA, Tech. Rep., 2007. [Online]. Available: https://api.semanticscholar.org/CorpusID

[44] K. Alzarooni, "Malware variant detection," Ph.D. dissertation, Dept. Comput. Sci., Univ. College London, London, U.K., 2012.

[45] M. Alonso, D. Andreu, R. Canal, S. Di Carlo, C. Chenet, J. Costa, A. Girones, D. Gizopoulos, V. Karakostas, B. Otero, G. Papadimitriou, E. Rodríguez, and A. Savino, "Validation, verification, and testing (VVT) of future RISC-V powered cloud infrastructures: The vitamin-V horizon Europe project perspective," in *Proc. IEEE Eur. Test Symp. (ETS)*, May 2023, pp. 1–6.

[46] H. Sayadi, N. Patel, S. M. P. D., A. Sasan, S. Rafatirad, and H. Homayoun, "Ensemble learning for effective run-time hardware-based malware detection: A comprehensive analysis and classification," in *Proc. 55th ACM/ESDA/IEEE Design Autom. Conf. (DAC)*, Jun. 2018, pp. 1–6.

[47] H. Sayadi, H. M. Makrani, S. M. Pudukotai Dinakarrao, T. Mohsenin, A. Sasan, S. Rafatirad, and H. Homayoun, "2SMaRT: A two-stage machine learning-based approach for run-time specialized hardware-assisted malware detection," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2019, pp. 728–733.

[48] N. Patel, A. Sasan, and H. Homayoun, "Analyzing hardware based malware detectors," in *Proc. 54th ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2017, pp. 1–6.

[49] H. Sayadi, M. Aliasgari, F. Aydin, S. Potluri, A. Aysu, J. Edmonds, and S. Tehranipoor, "Towards AI-enabled hardware security: Challenges and opportunities," in *Proc. IEEE 28th Int. Symp. On-Line Test. Robust Syst. Design (IOLTS)*, Sep. 2022, pp. 1–10.

[50] S. Dutto, A. Savino, and S. Di Carlo, "Exploring deep learning for in-field fault detection in microprocessors," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Feb. 2021, pp. 1456–1459.

[51] G. Torres and C. Liu, "Where's Waldo? Identifying anomalous behavior of data-only attacks using hardware features," in *Proc. 19th ACM Int. Conf. Comput. Frontiers*. New York, NY, USA: Association for Computing Machinery, May 2022, pp. 75–84, doi: 10.1145/3528416.3530226.

[52] D. Kasap, A. Carpegna, A. Savino, and S. Di Carlo, "Micro-architectural features as soft-error markers in embedded safety-critical systems: Preliminary study," in *Proc. IEEE Eur. Test Symp. (ETS)*, May 2023, pp. 1–5.

[53] B. Zhou, A. Gupta, R. Jahanshahi, M. Egele, and A. Joshi, "Hardware performance counters can detect malware: Myth or fact?" in *Proc. Asia Conf. Comput. Commun. Secur.* New York, NY, USA: Association for Computing Machinery, May 2018, pp. 457–468, doi: 10.1145/3196494.3196515.

[54] B. Zhou, A. Gupta, R. Jahanshahi, M. Egele, and A. Joshi, "A cautionary tale about detecting malware using hardware performance counters and machine learning," *IEEE Des. Test. Comput.*, vol. 38, no. 3, pp. 39–50, Jun. 2021.

[55] M. Botacin and A. Grégio, "Why we need a theory of maliciousness: Hardware performance counters in security," in *Information Security*, W. Susilo, X. Chen, F. Guo, Y. Zhang, and R. Intan, Eds. Cham, Switzerland: Springer, 2022, pp. 381–389.

[56] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors," *ACM SIGARCH Comput. Archit. News*, vol. 42, no. 3, pp. 361–372, Jun. 2014, doi: 10.1145/2678373.2665726.

[57] O. Mutlu and J. S. Kim, "RowHammer: A retrospective," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 8, pp. 1555–1571, Aug. 2020, doi: 10.1109/TCAD.2019.2915318. https://doi.org/10.1109/TCAD.2019.2915318

[58] C. Li and J.-L. Gaudiot, "Detecting malicious attacks exploiting hardware vulnerabilities using performance counters," in *Proc. IEEE 43rd Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, vol. 1, Jul. 2019, pp. 588–597.

[59] X. Wang and J. Backer, "SIGDROP: Signature-based ROP detection using hardware performance counters," 2016, *arXiv:1609.02667*.

[60] NIST. (Jan. 31, 2024). *National Vulnerability Database: CVE-2016–5195 Detail*. [Online]. Available: https://nvd.nist.gov/vuln/detail/cve-2016-5195

[61] V. M. Weaver and S. A. McKee, "Can hardware performance counters be trusted?" in *Proc. IEEE Int. Symp. Workload Characterization*, Oct. 2008, pp. 141–150.

[62] A. Carelli, A. Vallero, and S. D. Carlo, "Shielding performance monitor counters: A double edged weapon for safety and security," in *Proc. IEEE 24th Int. Symp. On-Line Test. Robust Syst. Design (IOLTS)*, Jul. 2018, pp. 269–274.

[63] A. Carelli, A. Vallero, and S. Di Carlo, "Performance monitor counters: Interplay between safety and security in complex cyber-physical systems," *IEEE Trans. Device Mater. Rel.*, vol. 19, no. 1, pp. 73–83, Mar. 2019.

[64] V. M. Weaver, D. Terpstra, and S. Moore, "Non-determinism and overcount on modern hardware performance counter implementations," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Apr. 2013, pp. 215–224.

[65] 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.

[66] J. Barrera, L. Kosmidis, H. Tabani, E. Mezzetti, J. Abella, M. Fernandez, G. Bernat, and F. J. Cazorla, "On the reliability of hardware event monitors in MPSoCs for critical domains," in *Proc. 35th Annu. ACM Symp. Appl. Comput.* New York, NY, USA: Association for Computing Machinery, Mar. 2020, pp. 580–589, doi: 10.1145/3341105.3373955.

[67] S. P. Kadiyala, P. Jadhav, S.-K. Lam, and T. Srikanthan, "Hardware performance counter-based fine-grained malware detection," *ACM Trans. Embedded Comput. Syst.*, vol. 19, no. 5, pp. 1–17, Sep. 2020, doi: 10.1145/3403943.

[68] M. Ritter, A. Tarraf, A. Geiß, N. Daoud, B. Mohr, and F. Wolf, "Conquering noise with hardware counters on HPC systems," in *Proc. IEEE/ACM Workshop Program. Perform. Visualizat. Tools (ProTools)*, Nov. 2022, pp. 1–10.

[69] M. A. Sasongko, M. Chabbi, P. H. J. Kelly, and D. Unat, "Precise event sampling on AMD versus intel: Quantitative and qualitative comparison," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 5, pp. 1594–1608, May 2023.

[70] B. Sprunt, "The basics of performance-monitoring hardware," *IEEE Micro*, vol. 22, no. 4, pp. 64–71, Jul. 2002.

[71] N. C. Doyle, E. Matthews, G. Holland, A. Fedorova, and L. Shannon, "Performance impacts and limitations of hardware memory access trace collection," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 506–511.

[72] H. M. Abdulwahab, S. Ajitha, and M. A. N. Saif, "Feature selection techniques in the context of big data: Taxonomy and analysis," *Appl. Intell.*, vol. 52, no. 12, pp. 13568–13613, Sep. 2022, doi: 10.1007/s10489-021-03118-3.

[73] G. Chandrashekar and F. Sahin, "A survey on feature selection methods," *Comput. Electr. Eng.*, vol. 40, no. 1, pp. 16–28, Jan. 2014, doi: 10.1016/j.compeleceng.2013.11.024.

[74] S. Browne, C. Deane, G. Ho, and P. Mucci, "PAPI: A portable interface to hardware performance counters," in *Proc. Dept. Defense HPCMP Users Group Conf.*, 1999, pp. 1–8.

[75] I. Molnar and T. Gleixner. (Jul. 5, 2023). *Performance Counters for Linux*. [Online]. Available: https://lwn.net/Articles/337493

[76] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011, doi: 10.1145/2024716.2024718.

[77] N. Bruschi, G. Haugou, G. Tagliavini, F. Conti, L. Benini, and D. Rossi, "GVSoC: A highly configurable, fast and accurate full-platform simulator for RISC-V based IoT processors," in *Proc. IEEE 39th Int. Conf. Comput. Design (ICCD)*, Benin, Oct. 2021, pp. 409–416.

[78] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[79] Y. Gao, H. M. Makrani, M. Aliasgari, A. Rezaei, J. Lin, H. Homayoun, and H. Sayadi, "Adaptive-HMD: Accurate and cost-efficient machine learning-driven malware detection using microarchitectural events," in *Proc. IEEE 27th Int. Symp. On-Line Test. Robust Syst. Design (IOLTS)*, Jun. 2021, pp. 1–7.
[80] H. Sayadi, Y. Gao, H. M. Makrani, J. Lin, P. C. Costa, S. Rafatirad, and H. Homayoun, "Towards accurate run-time hardware-assisted stealthy malware detection: A lightweight, yet effective time series CNN-based approach," *Cryptography*, vol. 5, no. 4, p. 28, Oct. 2021. [Online]. Available: https://www.mdpi.com/2410-387X/5/4/28
[81] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, 2nd ed. Hoboken, NJ, USA: Wiley, 2000.
[82] K. Pearson, "Note on regression and inheritance in the case of two parents," *Proc. Roy. Soc. London*, vol. 58, pp. 240–242, Jan. 1895.
[83] H. Peng, F. Long, and C. Ding, "Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 8, pp. 1226–1238, Aug. 2005.
[84] B. Singh, D. Evtyushkin, J. Elwell, R. Riley, and I. Cervesato, "On the detection of kernel-level rootkits using hardware performance counters," in *Proc. ACM Asia Conf. Comput. Commun. Secur.* New York, NY, USA: Association for Computing Machinery, Apr. 2017, pp. 483–493, doi: 10.1145/3052973.3052999.
[85] A. Kwan. (Jan. 31, 2024). *Malware Detection at the Microarchitecture Level Using Machine Learning Techniques*. [Online]. Available: https://scholarworks.calstate.edu/downloads/mk61rp641
[86] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Nov. 2011.
[87] E. Frank, M. A. Hall, G. Holmes, R. Kirkby, B. Pfahringer, and I. H. Witten, "Weka—A machine learning workbench for data mining," in *Data Mining and Knowledge Discovery Handbook*. Berlin, Springer: Springer, 2005, pp. 1305–1314. [Online]. Available: http://researchcommons.waikato.ac.nz/handle/10289/1497
[88] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 1–58, Jul. 2009, doi: 10.1145/1541880.1541882.
[89] H. Sayadi, N. Patel, A. Sasan, and H. Homayoun, "Machine learning-based approaches for energy-efficiency prediction and scheduling in composite cores architectures," in *Proc. IEEE Int. Conf. Comput. Design (ICCD)*, Nov. 2017, pp. 129–136.
[90] Chronicle. (Jun. 29, 2023). *Virustotal*. [Online]. Available: https://www.virustotal.com/gui/home/upload
[91] Y. Qiao, X. Yun, and Y. Zhang, "How to automatically identify the homology of different malware," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, Aug. 2016, pp. 929–936.
[92] H. Hu, Z. L. Chua, S. Adrian, P. Saxena, and Z. Liang, "Automatic generation of data-oriented exploits," in *Proc. 24th USENIX Conf. Secur. Symp.*, 2015, pp. 177–192.
[93] C. Konstantinou, X. Wang, P. Krishnamurthy, F. Khorrami, M. Maniatakos, and R. Karri, "HPC-based malware detectors actually work: Transition to practice after a decade of research," *IEEE Des. Test. Comput.*, vol. 39, no. 4, pp. 23–32, Aug. 2022.
[94] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *Proc. 4th Annu. IEEE Int. Workshop Workload Characterization*, Dec. 2001, pp. 3–14.
[95] T. G. Dietterich, "Ensemble methods in machine learning," in *Multiple Classifier Systems*. Berlin, Germany: Springer, 2000, pp. 1–15.
[96] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, Aug. 1997. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S002200009791504X
[97] H. Sayadi, Y. Gao, H. H. Makrani, T. Mohsenin, A. Sasan, S. Rafatirad, J. Lin, and H. Homayoun, "StealthMiner: Specialized time series machine learning for run-time stealthy malware detection based on microarchitectural features," in *Proc. Great Lakes Symp. VLSI*. New York, NY, USA: Association for Computing Machinery, Sep. 2020, pp. 175–180, doi: 10.1145/3386263.3407585.

**CRISTIANO PEGORARO CHENET** (Student Member, IEEE) is currently pursuing the Ph.D. degree with the Department of Computer and Control Engineering, Politecnico di Torino. His current research interest includes cybersecurity.

**ALESSANDRO SAVINO** (Senior Member, IEEE) received the Ph.D. degree from Politecnico di Turin, Turin, Italy. He is currently an Associate Professor with the Department of Control and Computer Engineering, Politecnico di Torino. His research interests include approximate computing, reliability analysis, safety-critical systems, software-based self-test, operating systems, imaging algorithms, machine learning, and audio manipulation.

**STEFANO DI CARLO** (Senior Member, IEEE) received the M.S. and Ph.D. degrees in computer engineering and information technology from Politecnico di Torino, Italy, in 1999 and 2003, respectively. Since 2021, he has been a Full Professor with the Department of Control and Computer Engineering, Politecnico di Torino. His research interests include diverse range, including reliability analysis, FPGA design, memory testing, NVM memory reliability with ECC, design for testability, built-in self-test, fault simulation, and automatic test generation. With over 200 peer-reviewed publications in esteemed IEEE/ACM TRANSACTIONS, journals, and conference proceedings, he also contributes to the editorial board of top-tier journals. His involvement extends to serving on various organizing and program committees for major IEEE and ACM conferences and symposia. Notably, he is recognized as a Golden Core Member of the IEEE Computer Society and has received both Outstanding and Meritorious Awards for his volunteer efforts within the society.

Open Access funding provided by 'Politecnico di Torino' within the CRUI CARE Agreement