cryptography

MDPI

Article

Towards Accurate Run-Time Hardware-Assisted Stealthy Malware Detection: A Lightweight, yet Effective Time Series CNN-Based Approach †

Hossein Sayadi [1,*], Yifeng Gao [2], Hosein Mohammadi Makrani [3], Jessica Lin [4], Paulo Cesar Costa [5], Setareh Rafatirad [6] and Houman Homayoun [3]

[1] Department of Computer Engineering and Computer Science, California State University, Long Beach, CA 90840, USA
[2] Department of Computer Science, University of Texas Rio Grande Valley, McAllen, TX 78504, USA; yifeng.gao@utrgv.edu
[3] Department of Electrical and Computer Engineering, University of California, Davis, CA 95616, USA; hmakrani@ucdavis.edu (H.M.M.); hhomayoun@ucdavis.edu (H.H.)
[4] Department of Computer Science, George Mason University, Fairfax, VA 22030, USA; jessica@gmu.edu
[5] Department of Systems Engineering and Operations Research, George Mason University, Fairfax, VA 22030, USA; pcosta@gmu.edu
[6] Department of Computer Science, University of California, Davis, CA 95616, USA; srafatirad@ucdavis.edu
* Correspondence: hossein.sayadi@csulb.edu
† This work is an extended version of our paper published in Great Lakes Symposium on VLSI (GLSVLSI 2020).

Citation: Sayadi, H.; Gao, Y.; Mohammadi Makrani, H.; Lin, J.; Costa, P.C.; Rafatirad, S.; Homayoun, H. Towards Accurate Run-Time Hardware-Assisted Stealthy Malware Detection: A Lightweight, yet Effective Time Series CNN-Based Approach. *Cryptography* **2021**, *5*, 28. https://doi.org/10.3390/cryptography5040028

Academic Editor: Jim Plusquellic

Received: 3 October 2021
Accepted: 13 October 2021
Published: 17 October 2021

Abstract: According to recent security analysis reports, malicious software (a.k.a. malware) is rising at an alarming rate in numbers, complexity, and harmful purposes to compromise the security of modern computer systems. Recently, malware detection based on low-level hardware features (e.g., Hardware Performance Counters (HPCs) information) has emerged as an effective alternative solution to address the complexity and performance overheads of traditional software-based detection methods. Hardware-assisted Malware Detection (HMD) techniques depend on standard Machine Learning (ML) classifiers to detect signatures of malicious applications by monitoring built-in HPC registers during execution at run-time. Prior HMD methods though effective have limited their study on detecting malicious applications that are spawned as a separate thread during application execution, hence detecting *stealthy malware* patterns at run-time remains a critical challenge. Stealthy malware refers to harmful cyber attacks in which malicious code is hidden within benign applications and remains undetected by traditional malware detection approaches. In this paper, we first present a comprehensive review of recent advances in hardware-assisted malware detection studies that have used standard ML techniques to detect the malware signatures. Next, to address the challenge of stealthy malware detection at the processor's hardware level, we propose *StealthMiner*, a novel specialized time series machine learning-based approach to accurately detect stealthy malware trace at run-time using branch instructions, the most prominent HPC feature. *StealthMiner* is based on a lightweight time series Fully Convolutional Neural Network (FCN) model that automatically identifies potentially contaminated samples in HPC-based time series data and utilizes them to accurately recognize the trace of stealthy malware. Our analysis demonstrates that using state-of-the-art ML-based malware detection methods is not effective in detecting stealthy malware samples since the captured HPC data not only represents malware but also carries benign applications' microarchitectural data. The experimental results demonstrate that with the aid of our novel intelligent approach, stealthy malware can be detected at run-time with 94% detection performance on average with only one HPC feature, outperforming the detection performance of state-of-the-art HMD and general time series classification methods by up to 42% and 36%, respectively.

Keywords: machine learning; hardware-assisted malware detection; cybersecurity; stealthy malware; hardware performance counter; deep learning; time series classification

1. Introduction

Cybersecurity for the past decades has been in the front line of global attention as a critical threat to the security of computer systems and information technology infrastructure. With the growth and pervasiveness of cyber infrastructure in modern society and everyday life, secure computing has become critically important. Attackers are increasingly motivated and enabled to compromise software and computing hardware infrastructure. The increasing complexity of modern computing systems in different application domains has resulted in the emergence of new security vulnerabilities [1–4]. Cyber attackers make use of these vulnerabilities to compromise systems using sophisticated malicious activities. Malware, a broad term for any type of malicious software, is a piece of code designed by cyber attackers to infect the computing systems without the user consent serving for harmful purposes such as stealing sensitive information, unauthorized data access, and running intrusive programs on devices to perform Denial-of-Service (DoS) attack [5–7].

The rapid development of information technology has made malware a serious threat to computer systems. According to a recent McAfee Labs threat report more than 67 million new malware variants have been discovered in the first quarter of 2019 alone, a near 40% increase when compared to the last quarter of 2018 [8]. Given the exceedingly challenging task of detection of new variants of malicious applications, malware detection has become more crucial in modern computing systems. The recent proliferation of modern computing devices in mobile and Internet-of-Things (IoT) domains further exacerbates the impact of this pressing issue calling for effective malware detection solutions.

Traditional software-based malware detection techniques such as signature-based and semantic-based methods mostly impose significant computational overheads to the system and more importantly do not scale well [6,9–13]. Furthermore, they are unable to detect unknown threats making them unsuitable for devices with limited available computing and memory resources. The emergence of new malware threats requires patching or updating the software-based malware detection solutions (such as off-the-shelf anti-virus) that needs a vast amount of memory and hardware resources, which is not feasible for emerging computing systems especially in embedded mobile and IoT devices [3,14,15]. In addition, most of these advanced analysis techniques are architecture-dependent i.e., dependent on the underlying hardware, which makes the existing traditional malware detection techniques hard to import onto emerging embedded computing devices [4,14].

The arm-race between security analysts and malware developers is a never-ending battle with the complexity of malware changing as quickly as innovation grows. To address the inefficiency of conventional malware detection methods, Hardware-based Malware Detection (HMD) techniques, by employing low-level features captured by Hardware Performance Counters (HPCs), have emerged as a promising solution [16–18]. HMD methods reduce the latency of the detection process by order of magnitude with a small hardware overhead. In particular, recent studies have shown that malware can be differentiated from normal programs by classifying anomalies using Machine Learning (ML) techniques in low-level microarchitectural feature spaces captured by Hardware Performance Counters (HPCs) [4,14–22] to appropriately represent the application behavior. The HPCs are special-purpose registers implemented into modern microprocessors to capture the trace of hardware-related events such as executed instructions, suffered cache-misses, or mispredicted branches for a running program [16,18,23]. Current state-of-the-art HMD research focuses on the development and application of ML techniques on HPCs information for malware detection due to its ability to keep pace with malware evolution with less visibility to the attacker. Such methods result in much lower computational overheads since ML-based malware classifiers can be implemented in microprocessor hardware with significantly lower overhead as compared to the traditional software-based methods [14,18,24].

Malicious software attacks have continued to evolve in quantity and sophistication during the past decade. Due to the ever-increasing complexity of malware attacks and the financial motivations of attackers, malware trends are recently shifting towards *stealthy* attacks. A stealthy attack is a type of cybersecurity attack in which the malicious code

is hidden inside the benign application for performing harmful purposes [1,25–29]. An example of deploying stealthy malware is in document files in which the malware is capable of indirectly invoking other applications or libraries on the host as part of document editing. The main purpose of stealthy attacks is to remain undetected for a longer period of time in the computing system. The longer the threat remains undiscovered in the system, the more opportunity it has to compromise computers and/or steal information before a suitable detection mechanism can be deployed to protect against it. Stolfo et al. discovered a new type of stealthy threat referred to as embedded malware [25]. Under this threat, the attacker embeds the malicious code or file inside a benign file on the target host such that the benign and malicious applications are executed as a single thread on the system. It has been shown that traditional signature-based antivirus applications are unable to detect embedded malware even when the exact signature of malware is available in the detector database [1,25,28]. Embedded malicious software is potentially a serious and emerging security threat in which accurate and intelligent security countermeasures need to be developed to secure the computing systems against such attacks. In this paper, we primarily focus on detecting stealthy attacks where malicious code is hidden inside the benign program, both executed as a single thread on the target system, making the detection process more challenging.

Existing hardware-assisted malware detection methods have primarily assumed that the malware is spawned as a separate thread while executing on the target host [16–18,23,30–32]. However, in real-world scenarios malicious programs attempt to hide within a benign application to bypass the detection mechanisms. In HMD methods the HPC data is directly fed to a detector, therefore, for embedded malicious code hidden inside the benign application, the HPC data becomes contaminated as the collected events include the combined benign and malware microarchitectural events. In addition to the challenge of detecting embedded malware, prior studies on hardware-based malware detection performed a limited study on malware classification accounting for the availability of a large number (e.g., 8/16) and diverse type of HPCs accessed at a time. However, today's microprocessors, even in the high-performance domain have a limited number of HPC registers, due to the design complexity and cost of concurrent monitoring of microarchitectural events [33,34]. Due to deep pipelines, complex prefetchers, branch predictors, modern cache design, etc., adding HPCs is a challenge in terms of counting multiple events and maintaining counter accuracy simultaneously under speculative execution [35]. Even modern Intel Xeon architectures houses only 4–6 HPCs, compare to 2 in Pentium 4 and server-class Intel Atom processor, for the very same reason. For embedded mobile and IoT domains, the number of HPCs that can be accessed simultaneously is even smaller making the HMD with limited available HPCs more crucial.

In response to the aforementioned challenges, in this paper we present a specialized and lightweight time series machine learning-based approach called *StealthMiner*, to accurately and intelligently detect the embedded malicious patterns inside the benign programs using only one HPC feature (branch instruction). In particular, the main objective of this work is to effectively detect the malicious application embedded inside the benign program using the least number of microarchitectural events (only one HPC feature) in which the traditional machine learning-based solutions are unable to detect them with even 8–16 features. Using an effective feature reduction technique, we first identify the most prominent low-level feature for embedded malware detection. Next, we propose a lightweight scalable time series-based Fully Convolutional Neural Network (FCN) model that automatically identifies potentially contaminated samples in HPC-based time series and utilizes them to distinguish the stealthy malware from benign applications at run-time using only branch instructions as the most significant low-level feature. For the purpose of a thorough assessment, we evaluate the efficiency of *StealthMiner* across various performance evaluation metrics (F-measure, Precision, Recall, Accuracy, Area Under the Curve (AUC), Latency, and Model Size) against state-of-the-art solutions that depend on standard ML, deep learning, and time series classification models. The results of this work will assist

the designers in making effective architectural decisions to develop accurate and efficient intelligent countermeasures for securing modern computing systems against emerging malicious cyber attacks.

The remainder of this paper is organized as follows. The background and related work are described in Section 2. Section 3 presents the motivations and challenges of hardware-based stealthy malware detection using machine learning techniques. Next, the details of the proposed intelligent and customized hardware-assisted stealthy malware detection approach are described in Section 4. In addition, the comprehensive analysis of experimental results is presented in section 5. Finally, Section 6 concludes this study.

2. Related Work and Background

In this section, we categorize the related work and background on malware detection using low-level features into three subsections. First, an overview of hardware performance counter registers and their applications is presented. Second, we discuss the latest studies on hardware-based malware detection, and next, we focus on the prior works on the detection of embedded/stealthy malware.

2.1. Hardware Performance Counters

The complexity of today's computing systems has tremendously increased compared to the prior systems. Hierarchical cache subsystems and pipeline, non-uniform memory, simultaneous multithreading, and out-of-order execution have a significant impact on the performance of modern processors. Performance monitoring is an important characteristic of a microprocessor. Access to the performance monitoring hardware is typically offered in the form of hardware performance counters, special-purpose registers that are available in modern microprocessors which count different microarchitectural events [18,21,36].

The primary purpose of HPC registers is to analyze and tune the architectural level performance and power of running applications [37–39]. While HPCs are finding their way in various processor platforms from high-performance to low-power embedded processors and IoT devices, they are limited in the number of microarchitectural events that can be captured simultaneously. This is mainly due to the limited number of physical registers on the processor chip which are expensive to implement. A variety of processor platforms such as Intel, ARM, and AMD includes HPCs on its processors. The HPC registers are responsible to collect a myriad of low-level events such as cache access and misses, TLB hits and misses, branch mispredictions, etc. [18,22,30].

Performance counter registers are easily programmable across all platforms. Depending on the processor architecture, there are different numbers of HPC registers available [18,22,30]. For instance, the number of counter registers in the Intel Ivy-bridge and Intel Broadwell CPUs is limited to only four per processor core, meaning that only four HPCs can be captured simultaneously. In addition, Intel SandyBridge and Haswell architectures both have a total of 8 general-purpose counters per core. This limitation can be mitigated by multiplexing performance counters [19,24], but at the cost of accuracy degradation.

Recently, application areas of hardware performance counters are grown from mere performance analysis to detecting firmware modification in embedded systems [21], estimating system power consumption [40], and detection of malicious software at the hardware level [14,17,18,22] or even hardware trojans [41]. As a result, in this work, we have used the low-level information captured from HPC registers to identify the malicious patterns of applications by proposing effective and complexity-aware machine learning-based solutions addressing various critical challenges associated with run-time malware detection using microarchitectural features.

2.2. Hardware-Assisted Malware Detection

Demme et al. [16] proposed to deploy HPCs information for malware detection and demonstrated the effectiveness of using traditional ML models for hardware-based malware detection. They showed high detection accuracy results for Android malware

by applying complex ML algorithms including Artificial Neural Network (ANN) and K-Nearest Neighbour (KNN). Tang et al. [17] further proposed an HPC-based anomaly detection and discussed the feasibility of complex unsupervised learning on low-level features to detect buffer overflow attacks. Unsupervised algorithms though effective in detecting unknown malware, are complex incurring large overheads and requiring sophisticated analysis.

The works in [23,42] adopt sub-semantic features to recognize malicious software signatures using Logistic Regression (LR) and Neural Network classifiers. In addition, the authors presented modifications in the microprocessor pipeline to detect malware in a truly real-time setting that increases the cost and complexity of the proposed solution. Singh et al. [30] is another HMD work that deployed machine learning classifiers that are applied on synthetic HPC traces for detecting kernel rootkit attacks. They also used the Gain Ratio feature selection method to determine the most prominent features for each rootkit dataset. The authors achieve high prediction accuracy in detecting five self-developed synthetic rootkits models. Nevertheless, this work while important only focused on the detection of kernel rootkit attacks using a limited set of synthesis datasets.

The HMD study in [31] used logistic regression to classify malware into different types and trained a specialized classifier for detecting each class. They further used specialized ensemble learning to improve the accuracy of malware detectors. The research in [18] proposed ensemble learning techniques to facilitate run-time hardware-assisted malware detection and improved the performance of HMD by accounting for the impact of reducing the number of HPC features on the performance of malware detectors. In [24], a machine learning-based HMD is proposed that uses various traditional classifiers but requires 8 or more features to achieve high accuracy, which makes it less suitable for online malware detection.

The recent work in [22] proposed a two-stage machine learning-based approach for run-time malware detection in which in the first level classifies applications using a multiclass classification technique into either benign or one of the malware classes (Virus, Rootkit, Backdoor, and Trojan). In the second level, to have a high detection performance, the authors deploy a machine learning model that works best for each class of malware and further apply effective ensemble learning to enhance the performance of malware detection.

The work in [43] evaluated the suitability of HPCs for HMD. Though the presented experimental results in this study are mostly in favor of malware detection through HPCs, they claim that if HPC traces of malware and benign applications are similar, it is hard to detect malware. However, the robustness of malware detection highly depends on the type of classifier employed. In addition, it is likely to mislead the HMD methods, if the malware is crafted adversarially to perturb HPC patterns look similar to benign applications patterns, similar to adversarial attacks in CNNs for image processing [44]. However, no details on crafting such adversarial applications nor real-world samples are provided. Furthermore, this work has performed limited analysis on embedded malware and only shows that one benign program (Notepad++) infused with ransomware cannot be detected by traditional machine learning-based HMD without providing any effective solution to tackle the challenge of detecting stealthy malware.

Collectively, prior works on hardware-based malware detection have considered a threat model in which the malware is spawned as a separate thread within the application execution time. In particular, they have ignored assuming the malicious code embedded inside the benign programs which is a more threatening attack for today's computing systems. Furthermore, they used traditional ML-based algorithms with more than 4 HPC features to detect the malware with high accuracy. Our work is different, as it targets a more harmful attack, embedded malware, running within the same thread and execution binary of benign programs. It also proposes a lightweight machine learning-based approach which is capable of detecting patterns of embedded malware in the benign application at run-time using only one low-level microarchitectural feature.

2.3. Embedded Malware Detection

Stolfo et al. [25] was the first study that introduced a new type of stealthy threat referred to as embedded malware in which the attacker embeds the malicious code inside a benign file on the target host such that the benign and malicious applications are executed as a single thread on the target system. They further introduced a method referred to as file-print analysis in which they calculated 1-gram byte distribution of a file to identify the file type among PDF and DOC files. In the context of malware detection, their work focused on embedded malware detection only in PDF and DOC files. They deployed three different models for representing the benign distributions namely single centroid, multi-centroids, and exemplar files as centroids. Mahalanobis distance was calculated between the distributions obtained from these models and the n-gram distribution of a given file.

The work in [28] proposed static and run-time dynamic methods for detecting malware embedded in Word documents. For their static analysis method, they used an open-source application to decompose files and further produced a similarity score to carry out the final classification decision. They make use of a 5-gram model for benign and malicious documents based on their adequate memory and detection performance. Next, given the model, a "similarity" score was produced for the final classification decision. In their dynamic approach, they employed sandbox-based tests to evaluate OS crashes and unexpected changes in the system. However, it is acknowledged by the authors that their approach is not practical to be used as an independent embedded malware detection scheme.

The research in [45] used conditional Markov n-grams techniques to propose an anomaly detection scheme to detect embedded stealthy malware. The rationale for using this type of n-grams is that it offers a more meaningful representation of a file's statistical properties as compared with traditional n-grams techniques. To this aim, they first examine byte sequences in benign programs to show that benign programs' data generally exhibit a 1-st order dependence structure. Using this correlation, they model the conditional distributions as Markov n-grams. They deployed entropy rate, an information-theoretic metric, to quantify changes in Markov n-gram distributions of a file and showed that the entropy rate Markov n-grams obtains significantly disturbed at malware embedding locations indicating its robustness for embedded malware detection. Their results indicate that the proposed Markov n-gram detector offers higher detection accuracy and false-positive rates as compared with the prior work on embedded malware detection in [25].

2.4. Time Series Classification

Since our proposed method in this work considers the embedded HMD problem as a time series classification task, here, we briefly discuss the recent works on time series classification.

Time series classification approaches can be divided into two different types, shapelet-based [46] and bag-of-pattern-based [47]. The shapelet-based approach [46] attempts to find the subsequences of data that are the most discriminating of classes and deploys them to generate features for classification. These subsequences can be used to transform the original inseparable raw time series into a lower-dimensional space that is easier to classify. In this type of model, each original time series can be transformed to a distance feature vector by computing the closest match distance between the time series and each of the shapelets.

The work in [48] proposed an algorithm to approximately select high-quality shapelets by using symbolic representation of the subsequence. Following a similar idea, the work in [49] introduced an approach to approximately find qualified shapelets via variable-length time series motif. In recent works, Grabocka et al. [50] and Li et al. [51] introduced a learning framework and a genetic algorithm-based framework, respectively, to generate a shapelet to classify the time series. Furthermore, Hills et al. [52] proposed an approach

called Shapelet Transformation (ST) to classify time series and achieve very high accuracy. However, the complexity of these approaches is very costly.

On the other hand, bag-of-pattern-based approaches attempt to discretize time series into a bag of symbols and deploy the distribution information for classification. Senin et al. [53] used a discretization technique called Symbolic Aggregation Approximation (SAX) to convert the subsequent time-series data into a bag of symbols and deploys a histogram of the symbols to represent the time series. Instead of using SAX representation, Schafer et al. [54] introduced a Symbolic Fourier Approximation (SFA) based discretization approach to generate the representation.

Recently, several deep learning-based time series classification approaches are proposed [55–58]. These approaches often utilized ML techniques such as convolution neuron network or LSTM neuron network to extract the features from time series. However, these models often consist of a large number of parameters incurring significant overhead and computational complexity to the computer system.

The complexity of all work mentioned above are very costly which makes them unfit to be used computer systems especially for resource-constrained devices with limited performance and power requirements. Recently, Schäfer and Li etc.[59,60] proposed a series of scalable time series classification approaches that are significantly faster than traditional time series classification models [46,53,54]. s a result, to better evaluate and highlight the effectiveness of our proposed approach for embedded malware detection (described in Section 5), we compare *StealthMiner* with state-of-the-art ML-based HMD solutions as well as the most recent scalable time series classification method [60].

3. Motivations

In this section, we discuss the motivations and challenges for proposing effective machine learning-based solutions for run-time stealthy malware detection using low-level hardware features.

3.1. Challenge of Detecting Stealthy Malware

Figure 1 illustrates the challenge of detecting embedded malware. Figure 1a visualizes the complete benign and malware HPC data (described in detail in Section 4), when the malware is spawned as a separate thread, via t-distributed Stochastic Neighbor Embedding (t-SNE) algorithm [61], a widely used algorithm for visualizing high dimensional data. As seen, the marginal area between malware and benign programs is large when malware is spawned as a separate thread indicating that by using traditional ML models (prior works) the malware can be easily detected. However, the converted points of embedded malware data are mixed with each other in Figure 1b depicting the impact of embedding malicious code inside benign applications.

The figure highlights the challenge of stealthy malware detection indicating that due to the dense distribution of malware and benign applications features, traditional classification approaches are not able to achieve high accuracy in detecting embedded malware. As a case study, by applying the nearest neighbor classifier on both complete and embedded malware datasets, the classifier can achieve an accuracy of 90% in detecting the malware as a separate thread. However, the classifier can only achieve nearly 60% accuracy in stealthy malware detection tasks when the malicious code is hidden inside the normal program.

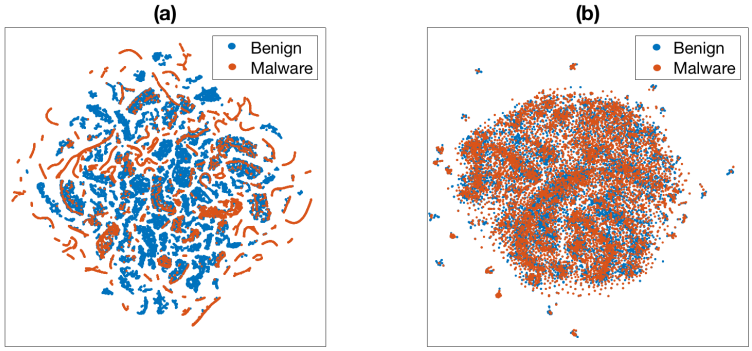

Figure 1. Visualizing the complete benign and malware dataset using the t-SNE algorithm: (**a**) malware spawned as a separate thread; (**b**) malware embedded inside benign applications.

3.2. Machine Learning for Hardware-Assisted Stealthy Malware Detection

As discussed, in this work, we intend to employ HPCs information to identify the behavior of running applications. As a case study to verify the suitability of using HPCs for ML-based malware detection, we executed malware and benign applications on an Intel Nehalem architecture-based system to observe the behavioral patterns of HPCs. The benign application is selected from MiBench [20] benchmark suite and the malware is a Backdoor application that can bypass the authentication process. The observed HPC traces of branch instructions for malware and benign applications are presented in Figure 2. The *X*-axis represents the time at which the HPC is monitored and the *Y*-axis represents the branch instruction HPC values.

The profiling trace shows that if two different programs are executed on a processor, they generate relatively different HPC traces, providing a unique opportunity to detect the behavior of the application. However, there exists an interesting observation in which if the malware is embedded inside a benign program from 0 ms to 1000 ms time intervals, there is a high possibility that the value of branch instructions for both benign and malware becomes equal which can mislead the traditional ML-based detectors in distinguishing the malicious behavior from benign applications. This highlights the importance and necessity of developing an effective intelligent approach as an alternative to traditional ML solutions to accurately detect the trace of embedded malware.

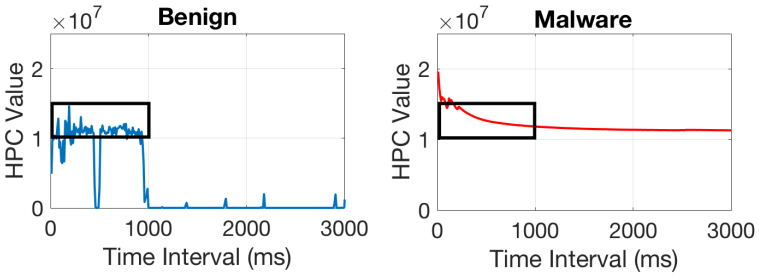

Figure 2. HPC traces of sample benign and malware (Backdoor) applications for branch-instruction HPC feature.

4. Proposed Intelligent Stealthy Malware Detection Framework

In this section, we describe the proposed machine learning-based approach for effective hardware-based stealthy malware detection. Figure 3 illustrates an overview of different steps for the proposed intelligent malware detection framework. As shown, it is comprised of different steps including data collection and feature extraction, feature reduction, and the proposed ML-based embedded malware detection approach (*StealthMiner*) each described in detail in the following subsections.

4.1. Experimental Setup and Data Acquisition

This section provides the details of the experimental setup and data collection process. The benign and malware applications are executed on an Intel Xeon X5550 machine (4 HPC registers available) running Ubuntu 14.04 with Linux 4.4 Kernel and HPC features are captured using *Perf* tool available under Linux at a sampling time of 10 ms. *Perf* provides rich generalized abstractions over hardware-specific capabilities. HPC-based profilers are currently built into almost every popular operating system. *Linux Perf* is a new implementation of performance counter support for Linux which is based on the Linux kernel subsystem *perf-event* and provides users a set of commands to analyze performance and trace data. It exploits *perf-event-open* function call in the background which can measure multiple events simultaneously. In our experiments, we executed more than 3500 benign and malware applications for data collection. Benign applications include real-world applications comprising MiBench [20] and SPEC2006 [62], Linux system programs, browsers, and text editors. Malware applications collected from virustotal and virusshare online repositories include Linux ELFs and scripts created to perform malicious activities and include 850 Backdoor, 640 Rootkit, and 1460 Trojan samples. The functionality of Backdoor applications is trying to provide remote access to the remote user (attacker) and facilitates information leakage; Rootkits provide the attackers with privilege access to modify the registers and authorized programs; and Trojans perform phishing of confidential information in the system.

In our experiments, the HPC information is collected by running applications in an isolated environment referred to as Linux Containers (LXC) [63]. LXC is chosen over other commonly available virtual platforms such as VMWare or VirtualBox since it provides access to actual performance counters data instead of emulating HPCs. To effectively address the non-determinism and overcounting issues of HPC registers in hardware-based security analysis discussed in recent works [43,64], we have extracted various hardware events available under *Perf* tool using static performance monitoring approach [34] where we can profile applications several times measuring different events each time. Furthermore, to ensure that running malware inside the Linux container does not contaminate the system's environment and also no contamination occurs in collected data due to the previous run of the program, the container is destroyed after each run.

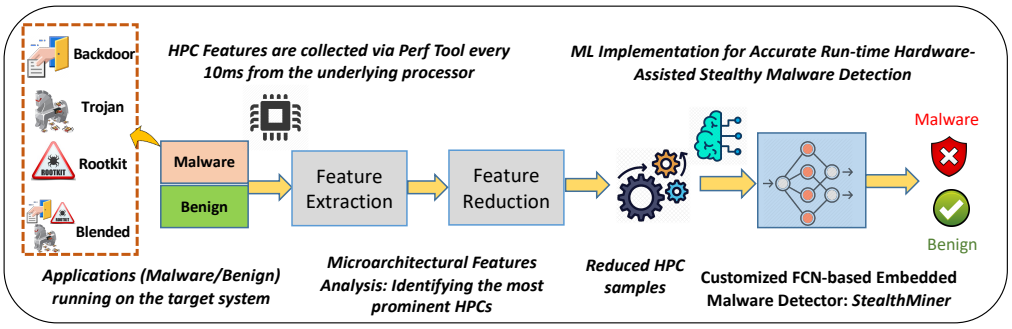

Figure 3. Overview of proposed hardware-assisted stealthy malware detection framework.

4.2. Feature Representation

Determining the most significant low-level features is an important step for effective HMD. As there exist numerous microarchitectural events (e.g., +100 in Intel Xeon), each of them representing a different functionality, collecting all features leads to high dimensional data. Furthermore, processing raw dataset involves computational complexity and induces delay. Hence, to perform an efficient run-time HMD with minimal overhead, we determine a minimal set of HPCs that can effectively represent the application behavior and are feasible to collect in a single run even on low-end processors with few HPCs. Therefore, instead of accounting for all captured features, irrelevant features need to be identified and removed using a feature reduction algorithm, and a subset of HPC events is selected that represents the most important features for classification. For the algorithmic selection

of features, we first use Correlation Attribute Evaluation to rank all captured features by calculating Pearson correlation between each attribute and class. The top features with the highest correlation coefficient value and their descriptions are shown in Table 1. These events have a mixture of branch-related events representing core behavior and cache-related events representing memory behavior. Next, we apply Principle Component Analysis (PCA) to find the best HPCs suited for training the ML-based malware detectors. PCA is a class of dimensionally reduction techniques that captures most of the data variation by rotating the original data to a new variable in a new dimension. We employ PCA to reduce the features and apply a hierarchical clustering technique to group similar features and identified the top 4 HPCs to capture the behavior of a specific class of malware. The feature reduction results indicate that the identified prominent 4 HPCs are the same across various classes of malware which includes branch instructions, cache references, branch misses, and node-stores.

Table 1. HPC events used for embedded malware detection and their description.

HPC Event	Description
Branch instructions	branch instructions retired
Branch-misses	branches mispredicted
Cache misses	last level cache misses
Cache-references	last level cache references
L1-dcache-load-misses	cache lines brought into L1 data cache
L1-dcache-loads	retired memory load operations
L1-dcache-stores	L1 data cache lines copied into DRAM
node-loads	successful load operations to DRAM
node-stores	successful store operations to DRAM
LLC-load-misses	cache lines brought into L3 cache from DRAM
LLC-loads	successful memory load operations in L3
iTLB-load-misses	misses in instruction TLB during load operations
Branch-loads	successful branches

The proposed time series-based detection approach, *StealthMiner*, using only the most significant HPC feature, branch instructions, can detect the embedded malware inside the benign application with high detection accuracy (will be discussed in detail in Section 5). Branch operations are one of the non-trivial microarchitectural events as most of the malware rely on branching operations for executing the malicious activity revealing the behavior of most malware programs. In addition, branch-related counters can be accessed even in most of the low-end embedded and IoT devices, therefore, making this type of microarchitectural event appealing to use for malware detection. Furthermore, it is hard to evade the branch instructions count due to the in-built exception the handler that notifies the user regarding the exception and terminates the program or it can result in long stalls, eventually leading to termination of ongoing executions.

4.3. Stealthy Malware Threat Models

The proposed intelligent hardware-assisted malware detection approach in this work is focused on the identification of a type of stealthy malware, referred to as an embedded malware attack which is a potential threat in today's computing systems that can hide itself within the running benign application on the system.

For modeling the embedded malware threats, we have considered persistent malicious attacks which occur once in the benign application with a notable amount of duration attempting to infect the system. For the purpose of thorough analysis, we deployed various malware types for embedding the malicious code inside the benign application including

Backdoor, Rootkit, Trojan, and Hybrid (Blended) attacks. For per-class embedded malware analysis, traces from one category of malware, are randomly embedded inside the benign applications and the proposed detection approach attempts to detect the malicious pattern.

Furthermore, the Hybrid threat combines the behavior of all classes of malware and hides them in the normal program. Persistent malicious codes are primarily a subset of Advanced Persistent Threat (APT) which is comprised of stealthy and continuous computer hacking processes, mostly crafted to perform specific malfunction activities. The purpose of persistent attacks is to place custom malicious code in the benign application and remain undetected for the longest possible period. Persistent malware signifies sophisticated techniques using malware to persistently exploit vulnerabilities in the systems usually targeting either private organizations, states, or both for business or political motives. The hybrid malware in our work represents a more harmful type of persistent threat in which the malicious samples are chosen from different classes of malware to achieve a more powerful attack functionality seeking to exploit more than one system vulnerability.

To create an embedded malware time series and model the real-world applications scenario, with capturing interval of 10 ms for HPC features monitoring, we consider 5 s. infected running application (benign application infected by embedded malware). For this study, 10,000 test experiments were conducted in which malware appeared at a random time during the run of a benign program. In our experiments, three different sets of data including training, validation, and testing sets are created for comprehensive evaluation of the *StealthMiner* approach. Each dataset contains 10,000 complete benign HPC time series and 10,000 embedded malware HPC time series. As the attacker can deploy unseen malware programs to attack the system, we create these three datasets with three groups of recorded malware HPC time series consisting of 33.3% for training, 33.3% for validation, and the remaining of whole recorded data for testing evaluation.

4.4. Overview of StealthMiner

As discussed, prior works on HMD mainly assumed that the malware is executed as a separate thread when infecting the computer system. This essentially means that the HPCs data captured at run-time inserted to the classifier belongs only to the malware program. In real-world applications, however, the malware can be embedded inside a benign application, rather than spawning as a separate thread, producing a more harmful attack. Therefore, the HPCs data captured at run-time in each interval could belong to both malware and benign application. As we will show in this work, this HPC data pollution could result in performance degradation of traditional ML classifiers. In response to this challenge, we propose *StealthMiner* malware detection framework which is based on a lightweight Fully Convolutional Neural Network (FCN)-based time-series classification. Primarily, the proposed FCN-based approach attempts to automatically identify potentially contaminated intervals in HPC-based time series at run-time and utilize them to distinguish the embedded malware from benign applications.

The overview of *StealthMiner* and its comparison with prior works is described in Figure 4. The network is a simplified version of neural network models inspired from previous general convolutional neural network-based time series classification models [55,56]. As shown in Figure 4a, our proposed solution in this work is based on the least number of HPC features and targets detecting stealthy attacks that have been ignored in prior studies on hardware-based malware detection. Furthermore, as seen in Figure 4b, the proposed FCN-based malware detector is created by stacking two 1-D convolution layers with 16 and 2 kernels, respectively. The size of the kernel in these two convolution layers is 2 and 3, respectively. These convolution layers aim at selecting the subsequence of the HPC time series for identifying the malware. Next, a global average pooling layer is applied to convert the output of the convolution layer into low dimension features. These features are then fed into a fully connected neural network to distinguish the embedded malware from benign applications.

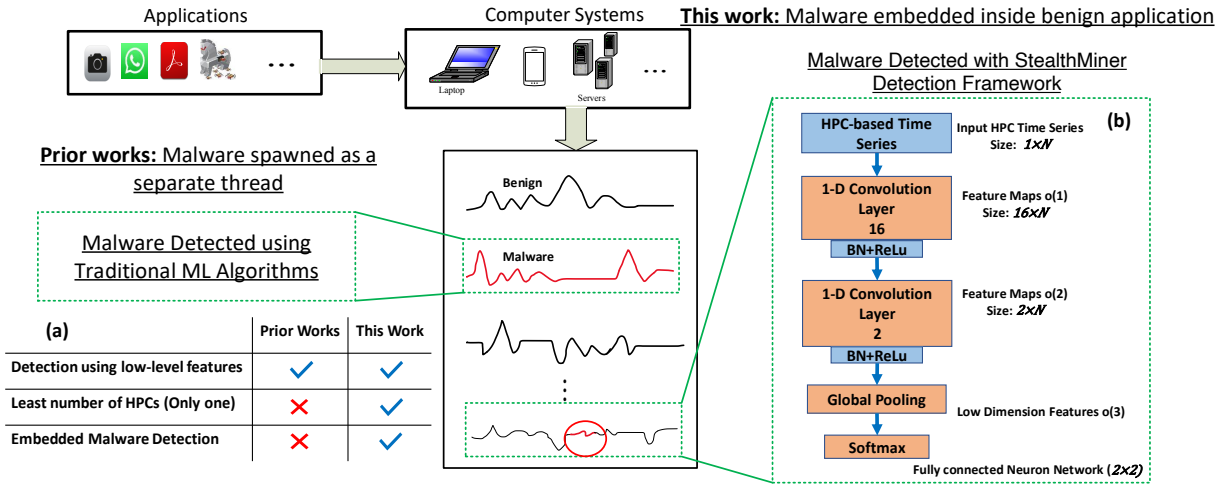

Figure 4. Overview of *StealthMiner*, Overview of StealthMiner, the proposed customized time series FCN-based approach for embedded malware detection (**b**) and its comparison with prior HMD works (**a**).

Concretely, given a time series of HPC features of $x = x_1, x_2, \ldots, x_N$, where N is the length of the time series in the first 1-D convolution layer, an output of kth kernel is computed by:

$$t_{i,k}^{(1)} = \sum_{j \in 1,2} w_{k,j,1} x_{i+j-1} + b_1 \tag{1}$$

where 2-d vector $[w_{k,1,1}, w_{k,2,1}] \in \mathbf{w}$ is the weight of kth kernel and $\mathbf{w} = \{w_{k,j,1} | k = 1, \ldots, 16, j = 1, 2\}$ is a 16×2 matrix that describes all weights of first layer. Given $t_k^{(1)} = [t_{1,k}^{(1)}, \ldots, t_{N,k}^{(1)}]$, a batch normalization function, $t_k^{(2)} = BN(t_k^{(1)})$, and a ReLu activation function, $o_k^{(1)} = ReLu(t_k^{(2)})$, are then applied. $BN(.)$ is a function which normalizes mean and variance of the $t_k^{(1)}$ to 0 and 1, respectively. Given an input vector x, $BN(.)$ can be written as below:

$$BN(t_{i,k}^{(1)}) = \frac{t_{i,k}^{(1)} - \mu_k}{\sigma_k} \tag{2}$$

where μ_k and σ_k is the mean and variance of vector across kth kernel. ReLu activation function is a nonlinear activation function that sets any negative value in input $t_k^{(2)}$ to 0:

$$Relu(t_k^{(2)}) = max(0, t_k^{(2)}) \tag{3}$$

The first layer output $o_k^{(1)}$ is a N dimension feature map generated from the kth kernel. We denote $o^{(1)} = [o_1^{(1)}, \ldots, o_{16}^{(1)}]$ as the output of the convolution layer. Intuitively, convolution layer converts original time series of length N into 16 different N dimensional feature maps capturing different potential local features that can be used to classify the input data [56].

The $o^{(1)}$ is then fed into next convolution layer with total number of kernels equal to 2. This layer summarizes $o^{(1)}$ into two different feature maps which can be computed via:

$$t_{i,k'}^{(3)} = \sum_{k=1}^{16} \sum_{j=1}^{3} w_{k',k,j,2} o_{i+j-1,k}^{(1)} + b_2 \tag{4}$$

where the weight of all kernels is a 3-d tensor $w_{k',k,j,2}$ of size $2 \times 16 \times 3$. For each $t_i^{(3)}$, $BN(.)$ and $ReLu(.)$ functions are further applied and four feature maps (denoted as $o^{(2)} = [o_1^{(2)}, o_2^{(2)}]$) are generated. Intuitively, stacking two convolution layers can increase the

accuracy of the framework and the ability of the model to detect complicated features which are not possible to be captured by a single convolution layer [56]. Note that any positive value inside the $o_1^{(2)}, o_2^{(2)}$ indicates the potential HPC intervals can be used to determine whether the input HPC time series contains embedded malware. Next, we conduct a global average pooling step to convert feature map $o^{(2)}$ into low dimension features. In particular, given a feature map of $o_k^{(2)} \in o^{(2)}$, we deploy the average value of all elements inside $o_k^{(2)}$ as the low dimension feature. As a result, this step converts $o^{(2)}$ into a 2-d vector (denoted as $o^{(3)}$).

Finally, $o^{(3)}$ is fed into a fully connected neural network with softmax activation function formulated below where a standard neural network layer is designed for our target classification task in detecting embedded malware:

$$o = Softmax(W^T o^{(3)} + b_3) \tag{5}$$

where $Softmax$ is the softmax activation function. It can be written as follows:

$$Softmax(x) = \frac{e^{x_i}}{\sum_{k=1}^{2} e^{x_k}} \tag{6}$$

The Equation (3) first converts $o^{(3)}$ into a new 2-d real value vector via linear transformation $W^T o^{(3)} + b_3$, where W is a 2×2 matrix and b_3 is a 2×1 vector. Next, all elements in the vector is mapped to [0,1] via $Softmax$ function. The final output is a 2-d vector $o = [o_1, o_2]$ which describes the possibility that the time series is benign or infected by malware (See Figure 5).

Suppose that we denote all the weights and the output of network as $\mathbf{\Theta}$ and $\mathbf{\Theta}(x) = [\mathbf{\Theta}_1(x), \mathbf{\Theta}_2(x)]$, respectively. Given a training dataset $\mathcal{D}$ and the network weights $\mathbf{\Theta}$, we update $\mathbf{\Theta}$ by minimizing the binary cross-entropy loss which can be computed by

$$L = \sum_{(\mathbf{x_i}, y_i) \in \mathcal{D}} -y_i \log(\mathbf{\Theta}_1(\mathbf{x}_i)) - (1 - y_i) \log(\mathbf{\Theta}_2(\mathbf{x}_i))) \tag{7}$$

where $\mathbf{x}_i$ and y_i is the HPC time series and the associated ground true label of the ith record in $\mathcal{D}$. And $y_i \in \{0, 1\}$ indicates whether the time series is benign or contains malware. Equation (7) can be minimized via a standard backpropagation algorithm, a widely used model for training various types of neural networks [55,56]. It primarily updates weights in the neural network by propagating the loss function value from the output to the input layer and iteratively minimizes the loss function for each layer via the gradient descent method. In this work, for each layer, the weights are optimized via Adam optimizer [65], a stochastic gradient descent method used to efficiently update weights of neural network.

To demonstrate the functionality of the StealthMiner approach in identifying the malware embedded inside the benign program, two illustrative case studies are presented Figure 5. As shown in Figure 5a, an HPC-based time series is an input to the classifier which contains an embedded rootkit malware (the embedded malware is highlighted in red). To identify the hidden malicious pattern, *StealthMiner* generates two feature maps $o_1^{(2)}$, $o_2^{(2)}$ via the proposed fully convolution neural network. The $o_1^{(2)}$ and $o_2^{(2)}$ are then categorized as a 2-d feature vector $o^{(3)}$ by calculating the simple average of all the values in the feature map. In the given example, $o^{(3)}$ is equal to [0.26, 0.32]. This 2-d feature is then fed into a fully connected neural network layer and the proposed detector analyzes the input HPC time series and attempts to find that whether the input trace contains an embedded malware or not in which in this case it successfully identifies the embedded malware with a significantly high probability (0.999). Similarly, when a benign HPC trace is fed into StealthMiner (as shown in Figure 5b), following the same process as the first example, the time series is converted into the 2-d feature vector ($[0.25, 0.1]$). Then, the 2-d vector is fed into the fully connected neural network layer and the network successfully identifies that it is a benign trace with a probability of 0.73.

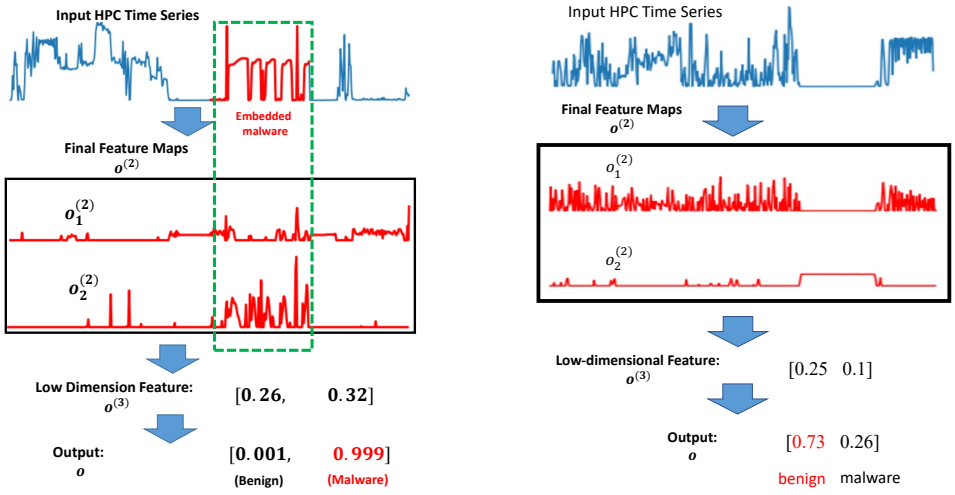

Figure 5. Illustrative case studies of *StealthMiner* in recognizing embedded malware via HPC time series traces. (**a**) Embedded malware is detected. (**b**) Input HPC trace is benign.

StealthMiner Implementation and Overhead: We implemented the proposed embedded malware detection framework via Pytorch deep learning library. For evaluating *StealthMiner* framework using performance metrics such as accuracy and F-measure (described in Section 5, the proposed detector determines whether the input time series contains embedded malware by computing the $argmax(o)$. For measuring the Area Under the Curve (AUC), we directly use the $output$ computed via Equation (3).

Different from existing neural network time series classification models proposed in prior works, the *StealthMiner* framework has a small total number of kernels and layers which dramatically reduces the number of parameters and the cost of detecting malware in the new HPC time series. For instance, in the latest neural network introduced by [55], to classify a time series the proposed solution needs more than 100,000 parameters. Hence, applying such heavyweight classification models to our embedded malware detection problem would significantly increase the overhead and complexity of our design, which certainly makes the solution impractical. In contrast, the *StealthMiner* framework only contains 200 parameters. Having a small number of parameters enhances the efficiency of the proposed ML-based malware detection solution highlighting the effectiveness and applicability of our proposed neural network-based approach to efficiently identify the embedded malware.

5. Experimental Results and Analysis

In this section, we evaluate the proposed embedded malware detection approach across different attack types and evaluation metrics with a comparison to existing techniques.

5.1. Performance Evaluation Criteria

In this work, the *StealthMiner* approach is evaluated using precision, recall (a.k.a. sensitivity), F-score, and detection accuracy (the overall rate of correctly classified samples). Typically, in binary classification techniques, the true positive *(tp)* metric represents the number of correctly classified positive samples, and the true negatives *(tn)* metric denotes the number of negative samples that are correctly classified. In addition, the false positive *(fp)* metric represents the incorrectly classified positive samples and the false negative *(fn)* metric is the number of negative samples that are incorrectly classified. The positive and negative terms show the success of the ML model, and the true and false terms specify if the classification is matched with the actual target class of application (malware or benign).

Accuracy (ACC): Detection accuracy of the ML classifier is evaluated by the ratio of correctly classified samples to a total number of samples. Notably, detection accuracy is an appropriate evaluation metric when the analyzed dataset is balanced. However, in real-world applications, it can be considered that the benign samples are far more than the malicious samples which could make the accuracy a less effective evaluation metric in an imbalanced dataset.

$$ACC = \frac{TP + TN}{TP + FP + TN + FN} \tag{8}$$

Precision (P): Precision metric is defined as the ratio of true positive samples to predicted positive samples represents the proportion of the sum of true positives versus the sum of positive instances which indicates the confidence level of malware detection. For instance, it is the probability for a positive sample to be classified correctly.

$$P = \frac{TP}{FP + TP} \tag{9}$$

Recall (R): Recall or True Positive Rate (TPR) or Sensitivity or hit rate is defined as the ratio of true positive samples to total positive samples and is also called the detection rate. It basically refers to the proportion of correctly identified positives or the rate of malware samples (i.e., positive instances) correctly classified by the classification model. The recall detection rate reflects the model's ability to recognize attacks which are calculated as follows:

$$TPR = \frac{TP}{TP + FN} \tag{10}$$

F-Measure (F): F-Measure or F-Score in machine learning is interpreted as a weighted average of the precision (P) and recall (R) that reaches its best value at 1 and worst at 0. F-Measure is a more comprehensive evaluation metric over accuracy (percentage of correctly classified samples) since it takes both the precision and the recall into consideration. More importantly, the F measure is also resilient to the class imbalance in the dataset which is the case in our experiments. Different measurements could be contradictory to each other. It is hard to meet with high precision and recall at the same time. We need to make a trade-off to balance them. Thus, F-measure i.e., F-score is often used to indicate detection performance. F Measure is calculated using the below equation:

$$FMeasure = \frac{2 \times (P \times R)}{P + R} \tag{11}$$

Area under the Curve (AUC): As the F-measure and accuracy are not the only metrics to determine the performance of the ML-based malware detectors, we also evaluate *StealthMiner* using Receiver Operating Characteristics (ROC) graphs. The ROC curve represents the fraction of true positives versus false positives for a binary classifier as the threshold changes. We further deploy the Area under the Curve (AUC) measure for ROC curves in the evaluation process which corresponds to the probability of correctly identifying malware and benign programs.and is more related to the robustness of the classifier. The robustness term is referred to how well the classifier distinguishes between binary malware and benign classes, for all possible threshold values. The AUC of the best possible classifier is equal to 1, meaning that we can find a discrimination threshold under which the classifier obtains 0% false positives and 100% true positives.

$$AUC = \int_0^1 TPR(x)dx = \int_0^1 P(A > \tau(x))dx \tag{12}$$

5.2. Evaluation of StealthMiner and Comparison to State-of-the-Arts

For the purpose of a comprehensive evaluation, we compared our proposed approach with recent general time series classification approaches, recent traditional machine

learning-based HMD techniques, and well-known deep learning algorithms that are widely used for cybersecurity and anomaly detection proposes.

5.2.1. *StealthMiner* vs. Traditional ML Models

We studied two general time series classification methods including a k-Nearest Neighbour (KNN) classifier, a classical time series classification method, and Bag-of-Pattern-Features (BOPF) [60] classifier, which is a recently proposed scalable time series classification approach. Given the input time series, the KNN classifier will assign the same class label to the input time series based on the most similar observed time series in the training set in which the similarity is measured by Euclidean distance.

As described before, Bag-of-Pattern-Features based time series classification approach is one of the recent fast time series classification algorithms that have a significantly low time and computational complexity compared with other existing time series classification approaches while maintaining a very high accuracy. As a result, for a comprehensive comparison of *StealthMiner* with state-of-the-arts, we implemented different ML-based HMD techniques and time series classification presented in recent prior works including JRip [18,24,32], J48 [18,24,32], Logistic Regression [23,24,31], KNN [16,32], and BOPF [60] that are representing the rule-based, decision tree, regression-based, and time series machine learning classifiers and have demonstrated high accuracy for detecting malware (spawned as a separate thread) in recent works.

Table 2 presents the evaluation results of malware detection for different classes of embedded malware for validation set. The results show that our proposed lightweight neural network-based solution can achieve average accuracy, precision, recall and F-score of nearly 0.9 across all types of experimented with embedded malware only by using the most prominent HPC feature (branch instructions). This makes the run-time detection of stealthy malware feasible which is primarily eliminating the need to execute applications multiple times to capture various low-level features suitable for HMD.

Table 2. Evaluation results of *StealthMiner* for the validation set.

Type	Precision	Recall	F-Score	Accuracy
Hybrid	0.85	0.89	0.88	0.87
Rookit	0.93	0.88	0.91	0.91
Trojan	0.91	0.87	0.89	0.89
Backdoor	0.88	0.94	0.91	0.91
Average	0.89	0.9	0.9	0.89

Figure 6 illustrates the ROC graphs of the proposed approach compare to state-of-the-art HMD and time series classification techniques. The correspondent AUC values for each embedded malware category are further presented in Table 3. A higher AUC value means that the ROC graph is closer to the optimal threshold and the classifier is performing better in terms of identifying the stealthy malware and classification of malware and benign applications. The ROC results clearly indicate the effectiveness of the proposed approach in this work as compared with prior ML-based malware detection and time series classification. As can be seen, our proposed approach, *StealthMiner*, achieves an average AUC value of 0.94 across all experimented categories of embedded malware. Furthermore, *StealthMiner* significantly outperforms the traditional ML algorithms used in recent HMD works, JRip, J48, and LR, by up to 0.48, and further outperforms tested time series classification approaches by up to 0.45 (for embedded Rootkit).

For the purpose of thorough analysis and comparison, Table 4 presents the accuracy, F-score, precision, and recall values testing evaluation results of proposed lightweight embedded malware detection approach in comparison with state-of-the-art works on hardware-based malware detection and time series classification. As seen, *StealthMiner* approach achieves highest accuracy and F-score for detection across all four different types

of embedded malware with the highest value of 0.93 for embedded Rootkit detection. Overall, the results indicate that *StealthMiner* performance is outperforming the state-of-the-art HMD methods (e.g., Logistic Regression) by up to 0.43 in accuracy and 0.44 in F-score highlighting the effectiveness of the proposed customized time series machine learning-based approach in detecting stealthy malware.

Table 3. AUC values of testing set results of *StealthMiner* vs. traditional ML-based detectors in prior works for detecting various embedded malware.

Attack Type	*StealthMiner*	JRIP	J48	LR	KNN	BOPF
Hybrid	**0.92**	0.64	0.62	0.53	0.6	0.7
Rookit	**0.98**	0.77	0.62	0.5	0.54	0.53
Trojan	**0.93**	0.85	0.69	0.57	0.65	0.79
Backdoor	**0.91**	0.73	0.54	0.51	0.6	0.68
Average	**0.94**	0.75	0.62	0.52	0.58	0.67

Table 4. Testing evaluation results of *StealthMiner* vs. traditional ML and time series classification techniques used in prior works.

Embedded Hybrid Malware				
Proposed vs. Prior Work	**Precision**	**Recall**	**F-Score**	**Accuracy**
StealthMiner	0.85	0.83	**0.86**	**0.89**
JRIP [18,24,32]	0.63	0.58	0.6	0.62
J48 [18,24,32]	0.63	0.57	0.6	0.62
LR [23,24,31]	0.52	0.49	0.51	0.52
BOPF [60]	0.97	0.41	0.58	0.7
KNN [16,32]	0.59	0.55	0.57	0.58
Embedded Rootkit Malware				
StealthMiner	0.95	0.9	**0.93**	**0.93**
JRIP [18,24,32]	0.81	0.68	0.74	0.76
J48 [18,24,32]	0.66	0.53	0.59	0.63
LR [23,24,31]	0.5	0.47	0.49	0.5
BOPF [60]	0.69	0.1	0.18	0.53
KNN [16,32]	0.55	0.46	0.5	0.54
Embedded Trojan Malware				
StealthMiner	0.92	0.82	**0.86**	**0.87**
JRIP [18,24,32]	0.84	0.78	0.81	0.82
J48 [18,24,32]	0.7	0.69	0.69	0.69
LR [23,24,31]	0.56	0.55	0.55	0.55
BOPF [60]	0.92	0.63	0.74	0.78
KNN [16,32]	0.63	0.72	0.67	0.65
Embedded Backdoor Malware				
StealthMiner	0.89	0.83	**0.86**	**0.86**
JRIP [18,24,32]	0.83	0.58	0.68	0.73
J48 [18,24,32]	0.59	0.31	0.41	0.55
LR [23,24,31]	0.5	0.43	0.46	0.51
BOPF [60]	0.93	0.38	0.54	0.68
KNN [16,32]	0.62	0.49	0.55	0.58

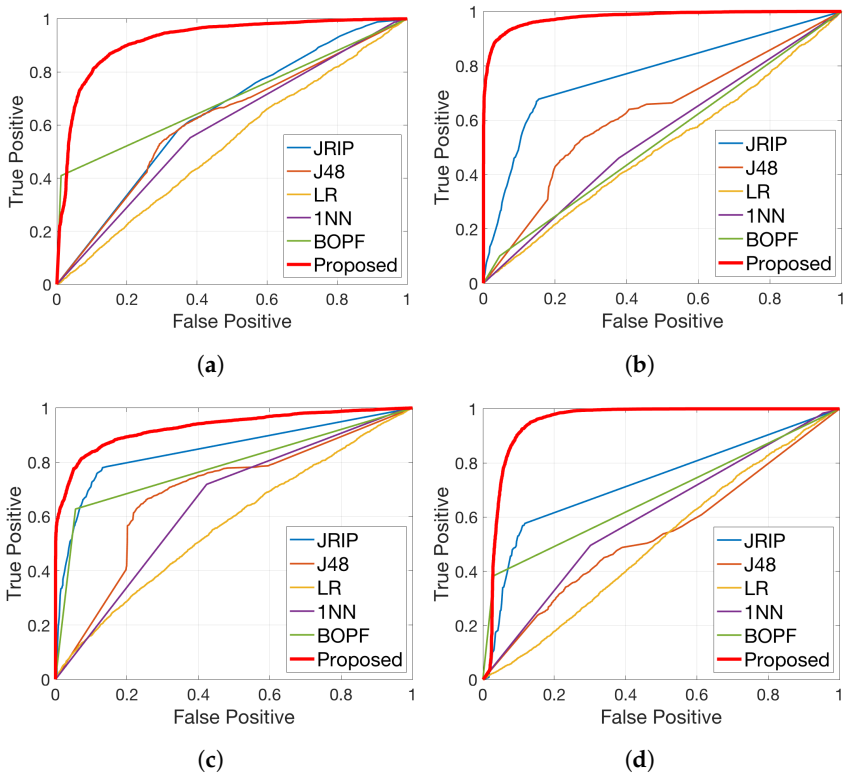

Figure 6. ROC graphs for detecting different classes of embedded malware. (**a**) Blended Malware. (**b**) Rootkit Malware. (**c**) Trojan Malware. (**d**) Backdoor Malware.

5.2.2. *StealthMiner* vs. Deep Learning Models used in Prior Works

Next, we compared the proposed framework with a series of deep learning-based time series classification models presented in previous works. To this aim, we compared *StealthMiner* with four popular deep learning-based time series classification models including Fully Convolutional Networks (FCN), Multilayer Perception (MLP), Deep Residual Neuron Network (ResNet), and Multi-Scale Convolutional Neural Networks (MCDCNN) as described below:

- *Fully Convolutional Networks (FCN):* [56] The heavy parameter version of the fully convolutional neural network model that consists of three convolution layers consisting of 128, 256, and 128 channels, respectively.
- *Multi-layer Perception (MLP):* [66] A classical multilayer perception deep learning Model that consists of three fully connected layers.
- *Deep Residual Neuron Network (ResNet):* [67] A deep convolutional neural network that consists of a skip-connection structure.
- *Multi-Scale Convolutional Neural Network (MCDCNN):* [68] A deep convolution neuron network that runs a Convolutional Neural Network with a different resolution of time series.

Performance Analysis: We first compared the detection performance of *StealthMiner* against the tested DL models. The results are shown in Table 5. As shown, compared with MLP and MCDCNN baselines, the proposed model achieves much higher performance. Compared with FCN and ResNet, *StealthMiner* has slightly decreased performance in detecting Hybrid, Backdoor, and Trojan malware. In embedded Rootkit malware detection tasks, *StealthMiner* achieves very similar F-measure and Accuracy against best baselines (0.93 vs. 0.94 and 0.93 vs. 0.95, respectively).

Table 5. Testing evaluation results of *StealthMiner* vs. Deep learning based approaches.

Embedded Hybrid Malware				
Proposed vs. Prior Work	**Precision**	**Recall**	**F-Score**	**Accuracy**
StealthMiner	0.85	0.83	0.86	0.89
FCN	0.97	0.91	0.94	0.94
MLP	0	0	0	0.5
ResNet	1	0.89	0.94	0.95
MCDNN	0	0	0	0.5
Embedded Rootkit Malware				
StealthMiner	0.95	0.90	0.93	0.93
FCN	1.00	0.78	0.88	0.89
MLP	0.50	1.00	0.67	0.50
ResNet	1.00	0.89	0.94	0.95
MCDNN	0.00	0.00	0.00	0.50
Embedded Trojan Malware				
StealthMiner	0.92	0.86	0.86	0.87
FCN	0.98	0.95	0.97	0.97
MLP	0.00	0.00	0.00	0.50
ResNet	1.00	0.83	0.91	0.92
MCDNN	0.50	1.00	0.66	0.50
Embedded Backdoor Malware				
StealthMiner	0.89	0.83	0.86	0.86
FCN	0.90	0.80	0.85	0.86
MLP	0.67	0.00	0.00	0.50
ResNet	1.00	0.94	0.97	0.97
MCDNN	0.00	0.00	0.00	0.50

Efficiency Analysis: We next compared the efficiency with all tested deep learning-based models. We analyzed the cost effectiveness of *StealthMiner* by considering two efficiency parameters representing the relative execution time (α_{time}) and the model size (α_{size}) (i.e., number of parameters required) of *StealthMiner* w.r.t to baseline deep learning algorithms. Specifically, we evaluated the performance by

$$\alpha_{time} = \frac{ExecutionTimeofBaselineModel}{ExecutionTimeofStealthMiner} \tag{13}$$

$$\alpha_{size} = \frac{ModelSizeofBaselineModel}{ModelSizeofStealthMiner} \tag{14}$$

Table 6 reports the execution time and model size results of *StealthMiner* as compared with other tested deep learning models for both execution time and the model size. According to the results, *StealthMiner* is significantly faster (by up to 6.52 times) than all the compared deep learning baseline models. This result indicates *StealthMiner* can result in much smaller computational latency that makes it an effective yet accurate solution for the online malware detection process. Moreover, *StealthMiner* contains up to 4375 times fewer parameters as compared with the most parameter-heavy baseline model. Thus, the lightweight characteristics of *StealthMiner* have dramatically reduced its complexity and memory footprints. Lastly, we demonstrated the efficiency (performance vs. cost) trade-off of each ML model. Specifically, the average F-measure (Accuracy) vs. Execution Time (Model Size) of *StealthMiner* and all the deep learning models are shown in Figure 7a–d. For instance, the Figure 7a indicates the trade-off between accuracy and execution time of the models in which *StealthMiner* achieves the best efficiency by delivering high detection rate while requiring significantly smaller execution time as compared to other models. Overall,

the results clearly highlight the effectiveness of our our proposed intelligent lightweight method, *StealthMiner*, in which it achieves a significantly better efficiency while maintaining a high detection rate with a very close accuracy and F-measure performance to the complex and heavyweight deep learning models.

Table 6. Execution time and model size results of *StealthMiner* as compared with deep learning models.

Model	Execution Time (s)	Model Size (# par.)	α_{time}	α_{size}
StealthMiner	0.95	172	×1	×1
FCN	4.0	265,986	×4.17	×1546
MLP	3.69	752,502	×3.85	×4375
ResNet	6.24	506,818	×6.52	×2946
MCDCNN	3.6	717,006	×3.76	×4168

Lastly, we analyze the advances, differences and limitations of our proposed intelligent solution as compared with prior works. To this aim, we compare the performance and efficiency characteristics of *StealthMiner* against three different types of learning models (deep learning classifier, classical ML classifier, and efficient time series classifier) for stealthy malware detection. A comparison between all the methods tested in this paper is shown in the Table 7. In the table, each column represents a model and each row represents an evaluation metric including performance (detection rate), Cost (Complexity and Latency), and efficiency (trade off between performance and cost). The × sign indicates the model is bad at a metric, ◯ indicates the model is good at this metric, and △ indicates the performance is good but slightly worse than ◯.

Table 7. Comparison of *StealthMiner* against baseline learning classifiers presented in prior studies.

Model	Deep Learning					Classical ML				Efficient TS
	StealthMiner	**FCN**	**MLP**	**ResNet**	**MCDNN**	**JRip**	**J48**	**LR**	**KNN**	**BOPF**
Performance	△	◯	×	◯	×	×	×	×	×	×
Cost	△	×	×	×	×	◯	◯	◯	◯	◯
Perf vs. Cost	◯	×	×	×	×	×	×	×	×	×

Comparing with the deep learning based models, *StealthMiner* has significantly fewer parameters and faster execution time. Since hardware-assisted malware detection has a strong requirement of efficiency, *StealthMiner* is more suitable for stealthy malware detection tasks compared with other deep learning models even with slightly lower detection performance. Moreover, as compared with classical machine learning classifiers and efficient time series classification approach, *StealthMiner* is more efficient in terms of the trade-off between performance and cost. We observe that the standard ML-based approaches have significantly worse malware detection performance compared with *StealthMiner* in our experiments across all four types of malware tested. Therefore, *StealthMiner* is also a more effective and balanced choice as compared with these methods when the computation cost is tolerable.

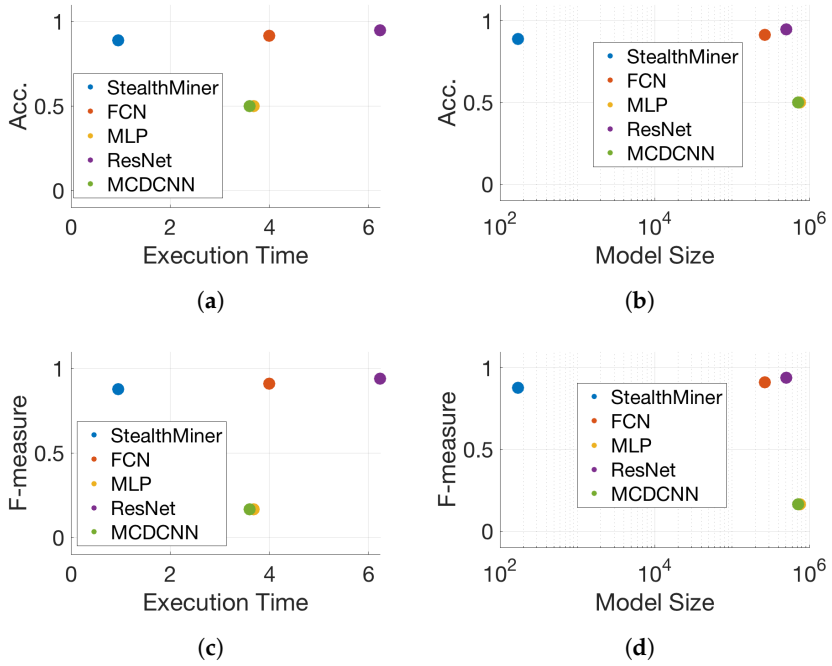

Figure 7. Efficiency analysis *StealthMiner* as compared with deep learning models. (**a**) Acc. vs. Execution Time. (**b**) Acc. vs. Model Size. (**c**) F-measure vs. Execution Time. (**d**) F-measure vs. Model Size.

6. Concluding Remarks and Future Directions

Malware detection at the hardware level has emerged as a promising solution to improve the security of computer systems. The existing works on Hardware-based Malware Detection (HMD) primarily assume that the malware is spawned as a separate thread while infecting the target system. However, detecting stealthy malware attacks, malicious code embedded in a benign application, at run-time is significantly a more challenging problem in today's computer systems, as the malware hides in the normal application execution. Embedded malware is a category of stealthy cybersecurity threats that allow malicious code to be hidden inside a benign application on the target computer system and remain undetected by traditional signature-based methods and commercial antivirus software. In hardware-based malware detection methods, when the HPC data is directly fed into a machine learning classifier, embedding malicious code inside the benign applications leads to contamination of HPC information, as the collected HPC features combine benign and malware microarchitectural events together.

In response, in this work we proposed *StealthMiner*, a lightweight time series-based Fully Convolutional Neural Network framework to effectively detect the embedded malware that is concealed inside the benign applications at run-time. Our novel intelligent approach, using only the most significant low-level feature, branch instructions, can detect the embedded malware with 94% detection performance (Area Under the Curve) on average at run-time outperforming the detection performance of state-of-the-art hardware-based malware detection methods by up to 42%. In addition, compared with the current state-of-the-art deep learning methods, *StealthMiner* is up to 6.52 times faster, and requires up to 4000 times less parameters. As the future directions of this work, we plan to explore the application of unsupervised anomaly detection and few-shot learning methods that could help train the detection model without requiring the ground truth with only a few or zero labels available. Moreover, as the next future line of our work we plan to examine the effectiveness of our proposed time series machine learning-based detector in resource-constrained mobile platforms. To this aim, we will expand our framework and experiments to ARM processor which is a widely used architecture in embedded systems and mobile applications. This direction could pave the way towards a more cost-effective run-time

stealthy malware detection in embedded devices with limited resources and computing power characteristics.

Author Contributions: Conceptualization, H.S. and H.H.; methodology, H.S. and Y.G.; software, H.S. and Y.G.; validation, H.S., Y.G., P.C.C. and H.H.; formal analysis, H.S. and Y.G.; investigation, H.S., Y.G. and H.M.M.; resources, H.S., J.L. and H.H.; data curation, H.S. and Y.G.; writing—original draft preparation, H.S. and Y.G.; writing—review and editing, H.M.M., P.C.C., J.L., S.R. and H.H.; visualization, H.S., Y.G. and H.M.M.; supervision, J.L., P.C.C., S.R. and H.H.; project administration, H.S., S.R. and H.H.; funding acquisition, H.H. and S.R. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded in part by NSF, grant number 1936836.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available in article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Sayadi, H.; Gao, Y.; Mohammadi Makrani, H.; Mohsenin, T.; Sasan, A.; Rafatirad, S.; Lin, J.; Homayoun, H. Stealthminer: Specialized time series machine learning for run-time stealthy malware detection based on microarchitectural features. In Proceedings of the 2020 on Great Lakes Symposium on VLSI (GLSVLSI'20), Virtual Event, China, 7–9 September 2020; pp. 175–180. [CrossRef]
2. Sayadi, H.; Wang, H.; Miari, T.; Makrani, H.M.; Aliasgari, M.; Rafatirad, S.; Homayoun, H. Recent advancements in microarchitectural security: Review of machine learning countermeasures. In Proceedings of the 2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS), Springfield, MA, USA, 9–12 August 2020; pp. 949–952.
3. Mosenia, A.; Jha, N.K. A comprehensive study of security of internet-of-things. *IEEE Trans. Emerg. Top. Comput.* **2016**, *5*, 586–602. [CrossRef]
4. Dinakarrao, S.M.P.; Guo, X.; Sayadi, H.; Nowzari, C.; Sasan, A.; Rafatirad, S.; Zhao, L.; Homayoun, H. Cognitive and scalable technique for securing IoT networks against malware epidemics. *IEEE Access* **2020**, *8*, 138508–138528. [CrossRef]
5. Bettany, A.; Halsey, M. What Is Malware? In *Windows Virus and Malware Troubleshooting*; Apress: Berkeley, CA, USA, 2017; pp. 1–8._1. [CrossRef]
6. Elhadi, A.A.E.; Maarof, M.A.; Osman, A.H. Malware detection based on hybrid signature behaviour application programming interface call graph. *Am. J. Appl. Sci.* **2012**, *9*, 283.
7. Jacob, G.; Debar, H.; Filiol, E. Behavioral detection of malware: From a survey towards an established taxonomy. *J. Comput. Virol.* **2008**, *4*, 251–266. [CrossRef]
8. McAfee Labs. Mcafee Labs Threats Report. 2019. Available online: https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-aug-2019.pdf (accessed on 13 October 2021).
9. Christodorescu, M.; Jha, S.; Kruegel, C. Mining Specifications of Malicious Behavior. In Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering (ESEC-FSE'07), New York, NY, USA, 3–7 September 2007; pp. 5–14.
10. Moser, A.; Kruegel, C.; Kirda, E. Limits of Static Analysis for Malware Detection. In Proceedings of the Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007), Miami Beach, FL, USA, 10–14 December 2007; pp. 421–430. [CrossRef]
11. Preda, M.D.; Christodorescu, M.; Jha, S.; Debray, S. A Semantics-based Approach to Malware Detection. In Proceedings of the 34th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'07), New York, NY, USA, 20–22 January 2007; pp. 377–388. [CrossRef]
12. Burguera, I.; Zurutuza, U.; Nadjm-Tehrani, S. Crowdroid: Behavior-based Malware Detection System for Android. In Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM'11), New York, NY, USA, 17 October 2011; pp. 15–26. [CrossRef]
13. Gu, G.; Porras, P.; Yegneswaran, V.; Fong, M.; Lee, W. BotHunter: Detecting Malware Infection through IDS-driven Dialog Correlation. In Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium, Berkeley, CA, USA, 6–10 August 2007; pp. 12:1–12:16.
14. Sayadi, H.; Makrani, H.; Randive, O.; Sai Manoj, P.D.; Rafatirad, S.; Homayoun, H. Customized Machine Learning-Based Hardware-Assisted Malware Detection in Embedded Devices. In Proceedings of the 17th IEEE International Conference On Trust, Security and Privacy in Computing and Communications (IEEE TrustCom-18), New York, NY, USA, 1–3 August 2018; pp. 1685–1688. [CrossRef]

15. Pudukotai Dinakarrao, S.M.; Sayadi, H.; Makrani, H.M.; Nowzari, C.; Rafatirad, S.; Homayoun, H. Lightweight Node-level Malware Detection and Network-level Malware Confinement in IoT Networks. In Proceedings of the 2019 Design, Automation Test in Europe Conference Exhibition (DATE), Florence, Italy, 25–29 March 2019; pp. 776–781. [CrossRef]
16. Demme, J.; Maycock, M.; Schmitz, J.; Tang, A.; Waksman, A.; Sethumadhavan, S.; Stolfo, S. On the Feasibility of Online Malware Detection with Performance Counters. In Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA'13), Tel-Aviv, Israel, 23–27 June 2013; pp. 559–570. [CrossRef]
17. Tang, A.; Sethumadhavan, S.; Stolfo, S.J. Unsupervised Anomaly-Based Malware Detection Using Hardware Features. In *Research in Attacks, Intrusions and Defenses*; Springer: Cham, Switzerland, 2014; pp. 109–129.
18. Sayadi, H.; Patel, N.; Sasan, A.; Rafatirad, S.; Homayoun, H. Ensemble Learning for Effective Run-time Hardware-based Malware Detection: A Comprehensive Analysis and Classification. In Proceedings of 55th ACM/ESDA/IEEE Design Automation Conference, San Francisco, CA, USA, 24–28 June 2018; pp. 1–6. [CrossRef]
19. Garcia-Serrano, A. Anomaly Detection for malware identification using Hardware Performance Counters. *arXiv* **2015**, arXiv:1508.07482.
20. Guthaus, M.R.; Pingenberg, J.S.; Austin, T.M.; Mudge, T.; Brown-MiBench, R.B. MiBench: A free, commercially representative embedded benchmark suite. In Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization, Austin, TX, USA, 2 December 2001; pp. 3–14. [CrossRef]
21. Wang, X.; Konstantinou, C.; Maniatakos, M.; Karri, R. Confirm ConFirm: Detecting firmware modifications in embedded systems using Hardware Performance Counters. In Proceedings of the 2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Austin, TX, USA, 2–6 November 2015; pp. 544–551. [CrossRef]
22. Sayadi, H.; Makrani, H.M.; Pudukotai Dinakarrao, S.M.; Mohsenin, T.; Sasan, A.; Rafatirad, S.; Homayoun, H. 2SMaRT: A Two-Stage Machine Learning-Based Approach for Run-Time Specialized Hardware-Assisted Malware Detection. In Proceedings of the 2019 Design, Automation Test in Europe Conference Exhibition (DATE'19), Grenoble, France, 25–29 March 2019; pp. 728–733. [CrossRef]
23. Ozsoy, M.; Donovick, C.; Gorelik, I.; Abu-Ghazaleh, N.; Ponomarev, D. Malware-aware processors: A framework for efficient online malware detection. In Proceedings of the 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA), Burlingame, CA, USA, 7–11 February 2015; pp. 651–661. [CrossRef]
24. Patel, N.; Sasan, A.; Homayoun, H. Analyzing Hardware Based Malware Detectors. In Proceedings of the 54th Annual Design Automation Conference (DAC '17), Austin, TX, USA, 18–22 June 2017; pp. 25:1–25:6. [CrossRef]
25. Stolfo, S.J.; Wang, K.; Li, W.J. Towards Stealthy Malware Detection. In *Malware Detection*; Christodorescu, M., Jha, S., Maughan, D., Song, D., Wang, C., Eds.; Springer US: Boston, MA, USA, 2007; pp. 231–249.
26. Rudd, E.M.; Rozsa, A.; Günther, M.; Boult, T.E. A Survey of Stealth Malware Attacks, Mitigation Measures, and Steps Toward Autonomous Open World Solutions. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 1145–1172. [CrossRef]
27. Zhang, H.; Yao, D.; Ramakrishnan, N. Detection of stealthy malware activities with traffic causality and scalable triggering relation discovery. In Proceedings of ACM Asia Conference on Computer and Communications Security (ASIA CCS'14), Kyoto, Japan, 4–6 June 2014; pp. 39–50.
28. Li, W.J.; Stolfo, S.; Stavrou, A.; Androulaki, E.; Keromytis, A.D. A Study of Malcode-Bearing Documents. In *Detection of Intrusions and Malware, and Vulnerability Assessment*; Springer: Berlin/Heidelberg, Germany, 2007; pp. 231–250.
29. Jiang, X.; Wang, X.; Xu, D. Stealthy Malware Detection Through Vmm-based "Out-of-the-box" Semantic View Reconstruction. In Proceedings of Conference on Computer and Communications Security (CCS'07), 31 October–2 November 2007; pp. 128–138.
30. Singh, B.; Evtyushkin, D.; Elwell, J.; Riley, R.; Cervesato, I. On the Detection of Kernel-Level Rootkits Using Hardware Performance Counters. In Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security (ASIA CCS'17), New York, NY, USA, 2–4 April 2017; pp. 483–493. [CrossRef]
31. Khasawneh, K.N.; Ozsoy, M.; Donovick, C.; Abu-Ghazaleh, N.; Ponomarev, D. Ensemble Learning for Low-Level Hardware-Supported Malware Detection. In *Research in Attacks, Intrusions, and Defenses*; Springer: Cham, Switzerland, pp. 3–25. Volume 9404._1. [CrossRef]
32. Bahador, M.B.; Abadi, M.; Tajoddin, A. HPCMalHunter: Behavioral malware detection using hardware performance counters and singular value decomposition. In Proceedings of the 2014 4th International Conference on Computer and Knowledge Engineering (ICCKE), Mashhad, Iran, 29–30 October 2014; pp. 703–708. [CrossRef]
33. Doyle, N.C.; Matthews, E.; Holl, G.; Fedorova, A.; Shannon, L. Performance impacts and limitations of hardware memory access trace collection. In Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE), Lausanne, Switzerland, 27–31 March 2017; pp. 506–511. [CrossRef]
34. Malone, C.; Zahran, M.; Karri, R. Are Hardware Performance Counters a Cost Effective Way for Integrity Checking of Programs. In Proceedings of the Sixth ACM Workshop on Scalable Trusted Computing, Chicago, IL, USA, 17 October 2011; pp. 71–76. [CrossRef]
35. Sprunt, B. The basics of performance-monitoring hardware. *IEEE Micro* **2002**, *22*, 64–71. [CrossRef]
36. Intel Performance Monitoring Unit. Available online: https://software.intel.com/en-us/articles/intel-performance-counter-monitor (accessed on 10 January 2020).

37. Sayadi, H.; Homayoun, H. Scheduling multithreaded applications onto heterogeneous composite cores architecture. In Proceedings of the 2017 Eighth International Green and Sustainable Computing Conference (IGSC), Orlando, FL, USA, 23–25 October 2017; pp. 1–8.
38. Malik, M.; Tullsen, D.M.; Homayoun, H. Co-locating and concurrent fine-tuning MapReduce applications on microservers for energy efficiency. In Proceedings of the 2017 IEEE International Symposium on Workload Characterization (IISWC), Seattle, WA, USA, 1–3 October 2017; pp. 22–31.
39. Makrani, H.M.; Sayadi, H.; Motwani, D.; Wang, H.; Rafatirad, S.; Homayoun, H. Energy-aware and machine learning-based resource provisioning of in-memory analytics on cloud. In Proceedings of the ACM Symposium on Cloud Computing, Carlsbad, CA, USA, 11–13 October 2018; pp. 517–517.
40. Bircher, W.L.; John, L.K. Complete System Power Estimation: A Trickle-Down Approach Based on Performance Events. In Proceedings of the 2007 IEEE International Symposium on Performance Analysis of Systems Software, San Jose, CA, USA, 25–27 April 2007; pp. 158–168. [CrossRef]
41. Elnaggar, R.; Chakrabarty, K.; Tahoori, M.B. Run-time hardware trojan detection using performance counters. In Proceedings of the 2017 IEEE International Test Conference (ITC), Fort Worth, TX, USA, 31 October–2 November 2017; pp. 1–10. [CrossRef]
42. Ozsoy, M.; Khasawneh, K.N.; Donovick, C.; Gorelik, I.; Abu-Ghazaleh, N.; Ponomarev, D. Hardware-Based Malware Detection Using Low-Level Architectural Features. *IEEE Trans. Comput.* **2016**, *65*, 3332–3344. [CrossRef]
43. Zhou, B.; Gupta, A.; Jahanshahi, R.; Egele, M.; Joshi, A. Hardware Performance Counters Can Detect Malware: Myth or Fact? In Proceedings of the 2018 on Asia Conference on Computer and Communications Security (ASIACCS'18), Incheon, Korea, 4 June 2018; pp. 457–468. [CrossRef]
44. Goodfellow, I.J.; Shlens, J.; Szegedy, C. Explaining and Harnessing Adversarial Examples. *arXiv* **2015**, arXiv:1412.6572.
45. Shafiq, M.Z.; Khayam, S.A.; Farooq, M. Embedded Malware Detection Using Markov n-Grams. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 88–107.
46. Ye, L.; Keogh, E. Time series shapelets: A new primitive for data mining. In Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'09), Paris, France, 28 June–1 July 2009; pp. 947–956.
47. Lin J, Li Y. Finding structural similarity in time series data using bag-of-patterns representation. In *Scientific and Statistical Database Management*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 461–477.
48. Rakthanmanon, T.; Keogh, E. Fast shapelets: A scalable algorithm for discovering time series shapelets. In Proceedings of the 2013 SIAM International Conference on Data Mining (SDM'13), Austin, TX, USA, 2–4 May 2013; pp. 668–676.
49. Wang, X.; Lin, J.; Senin, P.; Oates, T.; Gandhi, S.; Boedihardjo, A.P.; Chen, C.; Frankenstein, S. RPM: Representative Pattern Mining for Efficient Time Series Classification. In Proceedings of 19th International Conference on Extending Database Technology (EDBT'16), Bordeaux, France, 15–16 March 2016; pp. 185–196.
50. Grabocka, J.; Schilling, N.; Wistuba, M.; Schmidt-Thieme, L. Learning time-series shapelets. In Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'14), New York, NY, USA, 24–27 August 2014; pp. 392–401.
51. Li, X.; Lin, J. Evolving separating references for time series classification. In Proceedings of the 2018 SIAM International Conference on Data Mining (SDM'18), San Diego, CA, USA, 3–5 May 2018; pp. 243–251.
52. Hills, J.; Lines, J.; Baranauskas, E.; Mapp, J.; Bagnall, A. Classification of time series by shapelet transformation. *Data Min. Knowl. Discov.* **2014**, *28*, 851–881. [CrossRef]
53. Senin, P.; Malinchik, S. Sax-vsm: Interpretable time series classification using sax and vector space model. In Proceedings of the 2013 IEEE 13th International Conference on Data Mining (ICDM), Dallas, TX, USA, 7–10 December 2013; pp. 1175–1180.
54. Schäfer, P. The BOSS is concerned with time series classification in the presence of noise. *Data Min. Knowl. Discov.* **2015**, *29*, 1505–1530. [CrossRef]
55. Karim, F.; Majumdar, S.; Darabi, H.; Chen, S. LSTM fully convolutional networks for time series classification. *IEEE Access* **2018**, *6*, 1662–1669. [CrossRef]
56. Wang, Z.; Yan, W.; Oates, T. Time series classification from scratch with deep neural networks: A strong baseline. In Proceedings of the 2017 International Joint Conference on Neural Networks (IJCNN), Anchorage, AK, USA, 14–19 May 2017; pp. 1578–1585.
57. Zhang, X.; Gao, Y.; Lin, J.; Lu, C.T. Tapnet: Multivariate time series classification with attentional prototypical network. *Proc. AAAI Conf. Artif. Intell.* **2020**, *34*, 6845–6852. [CrossRef]
58. Yang, J.; Nguyen, M.N.; San, P.P.; Li, X.L.; Krishnaswamy, S. Deep convolutional neural networks on multichannel time series for human activity recognition. In Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI'15), Buenos Aires, Argentina, 25–31 July 2015; pp. 3995–4001.
59. Schäfer, P. Scalable time series classification. *Data Min. Knowl. Discov.* **2016**, *30*, 1273–1298. [CrossRef]
60. Li, X.; Lin, J. Linear Time Complexity Time Series Classification with Bag-of-Pattern-Features. In Proceedings of the 2017 IEEE International Conference on Data Mining (ICDM), New Orleans, LA, USA, 18–21 November 2017; pp. 277–286.
61. Van der Maaten, L.; Hinton, G. Visualizing data using t-SNE. *J. Mach. Learn. Res.* **2008**, *9*, 2579–2605.
62. Henning, J.L. SPEC CPU2006 Benchmark Descriptions. *SIGARCH Comput. Archit. News* **2006**, *34*, 1–17. [CrossRef]
63. Helsely, M. Lxc: Linux Container Tools. IBM Developer Works Technical Library. 2009. Available online: https://developer.ibm.com/tutorials/l-lxc-containers/ (accessed on 13 October 2021).

64. Das, S.; Werner, J.; Antonakakis, M.; Polychronakis, M.; Monrose, F. SoK: The challenges, pitfalls, and perils of using hardware performance counters for security. In Proceedings of the 2019 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 19–23 May 2019.
65. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.
66. Gardner, M.W.; Dorling, S. Artificial neural networks (the multilayer perceptron)—A review of applications in the atmospheric sciences. *Atmos. Environ.* **1998**, *32*, 2627–2636. [CrossRef]
67. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
68. Zheng, Y.; Liu, Q.; Chen, E.; Ge, Y.; Zhao, J.L. Time series classification using multi-channels deep convolutional neural networks. In Proceedings of the International Conference on Web-Age Information Management, Macau, China, 16–18 June 2014; pp. 298–310.