2SMaRT: A Two-Stage Machine Learning-Based Approach for Run-Time Specialized Hardware-Assisted Malware Detection

Hossein Sayadi[1], Hosein Mohammadi Makrani[1], Sai Manoj Pudukotai Dinakarrao[1],
Tinoosh Mohsenin[2], Avesta Sasan[1], Setareh Rafatirad[1], and Houman Homayoun[1]
[1]Department of Electrical and Computer Engineering, George Mason University
[2]Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County
[1]{hsayadi,hmohamm8,spudukot,asasan,srafatir,hhomayou}@gmu.edu, [2]{tinoosh}@umbc.edu

***Abstract*—Hardware-assisted Malware Detection (HMD) has emerged as a promising solution to improve the security of computer systems using Hardware Performance Counters (HPCs) information collected at run-time. While several recent studies proposed machine learning-based solutions to identify malware using HPCs, they rely on a large number of microarchitectural events to achieve high accuracy and detection rate. More importantly, they have largely overlooked complexity-effective prediction of malware classes at run-time. As we show in this work, the detection performance of malware classifiers is highly dependent on the number of available HPCs and varies significantly across classes of malware. The limited number of available HPCs in modern microprocessors that can be simultaneously captured makes run-time malware detection with high detection performance using existing solutions a challenging problem, as they require multiple runs of applications to collect a sufficient number of microarchitectural events. In response, in this paper, we first identify the most important HPCs for HMD using an effective feature reduction method. We then develop a specialized two-stage run-time HMD referred as 2SMaRT. 2SMaRT first classifies applications using a multiclass classification technique into either benign or one of the malware classes (Virus, Rootkit, Backdoor, and Trojan). In the second stage, to have a high detection performance, 2SMaRT deploys a machine learning model that works best for each class of malware. To realize an effective run-time solution that relies on only available HPCs, 2SMaRT is further customized using an ensemble learning technique to boost the performance of general malware detectors. The experimental results show that 2SMaRT using ensemble technique with just 4HPCs outperforms state-of-the-art classifiers with 8HPCs by up to 31.25% in terms of detection performance, on average across different classes of malware.**

Keywords*—*Run-Time Malware Detection, Hardware Performance Counters, Machine Learning

I. Introduction

Malware, short for malicious software, is a program developed by attackers with the intention of gaining access or causing damage to a computer system without the user agreement. Existing malware detection methods such as signature- and semantics-based anomaly detection techniques are software-based solutions that often incur significant computational overheads to the system [1, 2, 3, 4]. In response, the Hardware-assisted Malware Detection (HMD) techniques by employing underlying hardware-related information have shown promising results, reducing the latency of detection process by order of magnitude with small hardware costs [5, 6]. Recent works have demonstrated that malware can be differentiated from normal applications by classifying anomalies using Machine Learning (ML) techniques in low-level microarchitectural feature spaces captured by Hardware Performance Counters (HPCs) [5, 6, 7, 8, 9]. The HPCs are a set of special-purpose registers built into modern microprocessors to capture the trace of hardware-related events for a running program [8, 10, 11]. ML-based malware detectors can be implemented in microprocessor hardware with significantly low overhead as compared to the software-based methods, as detection inside the hardware is very fast within few clock cycles [2, 8].

Prior works on HMD performed limited study on malware classification; 1) ignoring the per-class of malware analysis, which can enhance the malware detection performance by employing specialized classifiers as shown in this work, and 2) accounting for the availability of a large number (e.g. 16 or 32) and diverse type of HPC features accessed at a time [2, 5, 6, 7, 8, 9, 12, 13, 14]. Modern processors, even in the high-performance domain have limited number of HPC registers (2 to 8), due to several reasons including the design complexity and cost of concurrent monitoring of microarchitectural events [15, 16]. 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 [8, 15]. Therefore, utilizing a variety of microarchitectural events, more than the number of available HPCs, to achieve high accuracy using general ML models presented in prior works requires executing the application multiple times, since the hardware can only count a small subset of events concurrently. This approach is not practical for run-time detection of malware.

While previous studies focus on one or few general ML classifiers and limited classes of malware, it is not clear which of the ML techniques deliver the best results across various metrics including detection rate and hardware design overheads as well as detection delay across various classes of malware. To better understand this, we first evaluate various ML classifiers from diverse range of ML models across different classes of malware. A quantitative comparison of our results indicate that there is no unique ML classifier that delivers the best results across all malware classes. In addition, the detection rate that varies across classes of malware is dependent on the number of available HPCs. This makes the detection and classification of malware highly dependent on the number of HPCs and the type of ML classifier used. Furthermore, not knowing the malware type ahead of time, makes it a challenge to deploy the right ML classifier for detection.

The objective of this work is to improve the performance of HMD for different classes of malware using the small number of HPC registers available in today's microprocessors that can be captured at run-time. We propose a two-stage machine learning-based run-time specialized malware detection, referred as *2SMaRT*, to not only distinguish the malware from benign applications, but also to identify the class of mal-

978-3-9819263-2-3/DATE19/©2019 EDAA

ware at run-time using proper low-level features. To quantify the effectiveness of 2SMaRT, we precisely compare malware detectors in terms of F measure (detection rate metric), robustness, performance (F measure×robustness), and hardware implementation overhead to determine the most suited ML classifiers per malware class for complexity-effective run-time hardware-assisted malware detection. The key contributions of this work are summarized as follows:

- To facilitate an efficient run-time HMD by choosing an optimal set of available HPCs, an effective feature reduction method is proposed to determine the most prominent microarchitectural events across various classes of malware used for detecting each malware.
- Given the high correlation of detection performance, ML classifier type, and class of malware, we propose 2SMaRT, a two-stage run-time specialized HMD framework which comprised of a multiclass classifier that predicts the malware class or benign applications in the first stage followed by complexity-effective specialized ML classifiers for efficient and highly accurate per-class malware detection.
- We further propose Boosted-HMD using an effective ensemble learning technique in the second stage of 2SMaRT to improve the performance of HMD when using a small number of features captured at run-time by the existing HPCs, eliminating the need to run an application multiple times.

II. Motivation and Background

A. Malware Detection using HPCs information

In this work, we use HPCs to collect execution traces for various microarchitectural events by executing malware and benign applications in an isolated environment. The profiling process shows that two different applications generate different HPC traces when executed on a processor, providing a unique opportunity to detect the behavior of running application (benign or malware). Fig. 1 illustrates the trace of two features, *branch instructions* and *branch misses*, for benign and malware applications. As can be seen, the malware traces are significantly different from benign applications for both features. This observation indicates that malware can be distinguished from normal applications using HPC information. Unlike prior studies, the analysis in this work is focused on run-time malware detection, as such we limit the number of microarchitectural events to 4, which is equal to the maximum number of HPCs that can be simultaneously read at run-time.

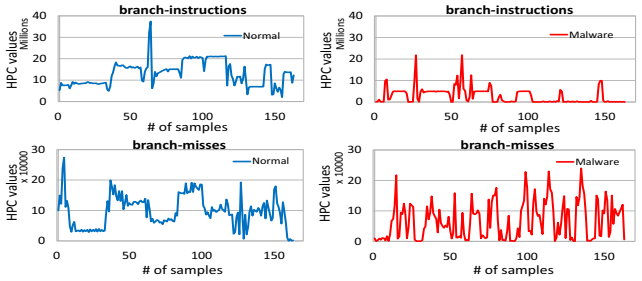

Fig. 1: HPC traces of branch-instructions and branch misses for sample malware and normal(benign) applications

B. The Need for Specialized Malware Detectors

As a case study to highlight the importance of per-class analysis of malware and specialized malware detection, Table I presents the ML classifiers that achieve highest detection rate in our experiments for different malware classes using various number of HPCs (will be discussed in details in section III). Given the reported results, it is observed that depending on the class of malware (Trojan, Virus, Rootkit, Backdoor) the type of ML classifier that performs best varies and there exists no ML classifier that performs best for all classes. In addition, the ML classifier that performs best also varies with the number of HPCs used. For instance, with 16 HPCs, MLP achieves best detection rate for Backdoor, but by reducing the number of HPCs to 4, OneR outperforms MLP. The disparity of optimal ML solutions across various classes of malware and varying number of HPC features implies the necessity of per-class malware analysis and developing effective specialized HMD for different classes of malware. Since each malware class has a different behavior, it allows a specialized detector to more effectively perform the classification. In this work, we implemented specialized detectors for four classes of malware including Backdoor, Virus, Rootkit, and Trojan.

TABLE I: ML classifiers with highest per-class detection accuracy

Malware Class	16HPCs	8HPCs	4HPCs
Trojan	JRip	JRip	MLP
Virus	OneR	J48	MLP
Rootkit	J48	J48	MLP
Backdoor	MLP	OneR	OneR

III. Proposed Malware Detection Framework

This section presents the details of our proposed run-time specialized hardware-assisted malware detection approach.

A. Experimental Setup and Data Collection

This section provides the details of the experimental setup and data collection process. The applications (both malware and benign) are executed on an Intel Xeon X5550 machine running Ubuntu 14.04 with Linux 4.4 Kernel. In order to extract the HPC information, we used *Perf* tool available under Linux. *Perf* provides rich generalized abstractions over hardware specific capabilities. It exploits *perf-event-open* function call in the background which can measure multiple events simultaneously. We executed more than 3000 benign and malware applications for HPC data collection. Benign applications include MiBench benchmark suite [17], Linux system programs, browsers, text editors, and word processor. For malware applications, Linux malware is collected from virustotal.com and virusshare.com. Malware applications include four classes of malware comprising 452 Backdoor, 350 Rootkit, 650 Virus, and 1169 Trojan samples.

Fig. 2 depicts the overview of the data collection process and proposed run-time HMD. It is primarily composed of various stages including feature extraction, feature reduction, and ML classifiers (general and ensemble) implementation for malware detection. In our experiments, HPC information is collected by executing all applications in Linux Containers (LXC) [18] which is an isolated environment providing access to actual performance counters data instead of emulating HPCs. We extracted 44 CPU events available under *Perf* tool. Since Intel Xeon has only 4 HPC registers available [19], we can only capture 4 events at a time. As a result, multiple runs are required to fully capture all events. We divide 44 events into 11 batches of 4 events and run each application 11 times at sampling time of 10ms to gather all microarchitectural events. Running malware inside the container can contaminate the environment which may affect subsequent data collection. To ensure that there is no contamination in collected data

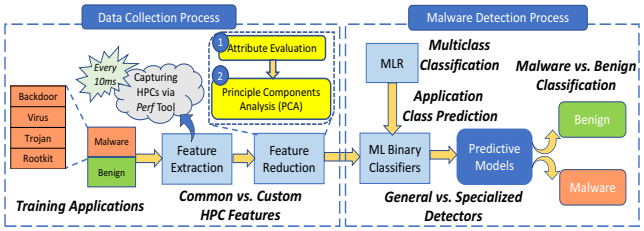

Fig. 2: Overview of the proposed HMD framework

due to the previous run, the container is destroyed after each run. After collecting microarchitectural events using *Perf*, we deploy WEKA tool [20] for evaluating the detection rate and performance of various ML classifiers. In order to validate each of the utilized ML classifiers, a standard 60%-40% dataset split for training and testing is followed.

B. Feature Reduction

Detecting malware using ML models requires representing programs at low-level features which leads to a high-dimensional data processing involved large computational overheads and complexity. Furthermore, incorporating irrelevant features would result in lower accuracy and performance for the classifiers [8, 21]. On the other hand, as we show in this work, microarchitectural events representing behavior of malware (and can be used to distinguished them from benign applications) varies across classes of malware. This poses two important research questions. First, which low-level features are relevant to be employed to detect and classify a particular class of malware? Second, how to perform feature reduction of collected data to alleviate unnecessary computational overheads? In order to detect malware at run-time with minimal overhead and avoid multiple runs, we intend to identify a minimal set of critical HPCs that can effectively represent the malware class behavior and are feasible to collect even on low-end processors with small number of HPCs in a single run. Therefore, instead of accounting for all captured features, irrelevant features need to be identified and removed using an effective feature reduction algorithm. A subset of HPC features is selected representing the most important features for classification. The selected features are then supplied to each ML-based malware detector. The detector attempts to find a correlation between the feature values and the application behavior to predict the benign or malware type.

We started from 44 performance counters features during training. As shown in Fig. 2, we first use Correlation Attribute Evaluation on our training set under WEKA to monitor the most 16 vital microarchitectural parameters to capture application characteristics. Next, we apply Principle Component Analysis (PCA) technique on the 16 extracted HPCs to determine the most prominent HPCs for capturing applications behavior and detecting malware during run-time. 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, commonly known as the principal components (PC). These new variables are uncorrelated to each other and are a linear combination of the original data. We employ PCA to project our original gathered features into a new dimensional space to determine the most important features along different PC dimensions. Using PCA, we reduced the features to 8 most significant ones to capture the behavior of specific malware type. Due to space limitations, we only show the top 8 features for each malware class in Table II. These features are included as input parameters in our detection and classification models. They include features representing pipeline front-end, pipeline back-end, cache subsystem, and main memory behaviors and are influential in the performance of standard applications.

TABLE II: Prominent top eight HPC features for each class of malware

HPC	Backdoor	Trojan	Virus	Rootkit
Common	*branch-inst* *cache-ref* *branch-miss* *node-st*	*branch-inst* *cache-ref* *branch-miss* *node-st*	*branch-inst* *cache-ref* *branch-miss* *node-st*	*branch-inst* *cache-ref* *branch-miss* *node-st*
Custom	branch-lds L1-icache-ld-miss LLC-ld-miss iTLB-ld-miss	cache-miss L1-icache-ld-miss LLC-ld-mis iTLB-ld-miss	LLC-lds L1-dcache-lds L1-dcache-st iTLB-ld-miss	cache-miss branch-lds LLC-ld-miss L1-dcache-st

C. 2SMaRT for Per-class Malware Detection

In this section, we describe the details of 2SMaRT, the proposed two-stage run-time specialized HMD approach.

Stage 1: Application Type Prediction. As each of the specialized detectors is trained to classify a different phenomenon (class of malware), they are each answering a different classification question. Initially, the system is unaware of existence of malware in the application, as such the use of specialized detectors cannot be effective. By analyzing the ML classifiers for malware detection across various classes, we observe (details are presented in Section IV) that the performance of malware detectors are highly correlated to the class of malware (Virus, Trojan, etc.) infecting the system. To address this challenge, we primarily convert the basic binary malware classification to a multiclass problem, i.e., with more than two possible discrete outcomes. For this purpose, we propose to predict the behavior of the application (benign or a particular malware class) using a Multinomial Logistic Regression (MLR) technique shown in first stage of Fig. 3.

The MLR classifier is a generalized linear model that predicts the probability of a discrete set of outcomes of categorically distributed dependent variable, given a set of independent variables which makes it a suitable classifier for predicting the class of applications. MLR is basically an extension of binary logistic regression that allows for more than two categories of the outcome variable. In this work, the output of MLR is corresponding to the set of feasible classes of applications including 5 individual classes, one for "benign" program and 4 malware classes namely "Virus", "Trojan", "Rootkit", and "Backdoor" classifying the four analyzed malware types. The MLR model is trained using extensive set of HPCs data captured by running various benign and malware programs. The inputs to the MLR consists of the top 4 low-level features shown in Table II. During run-time, the probability of each class of application being executed is calculated and the MLR classifier then selects the class that achieves the highest probability. The evaluation results of the proposed MLR show that while the detection rate for the MLR using 16 HPCs is shown to be 83%, lowering the number of HPCs to the 4 top HPCs does not reduce the accuracy rate of the classifier noticeably and results in accuracy of close to 80%. As a result, using only 4 top HPCs the MLR model can predict the type of running application.

Common vs. Custom Features. As can be seen in Table II, after the feature reduction, 4 out of the 8 identified microarchitectural events are the same across various classes of malware. These microarchitectural events are referred as

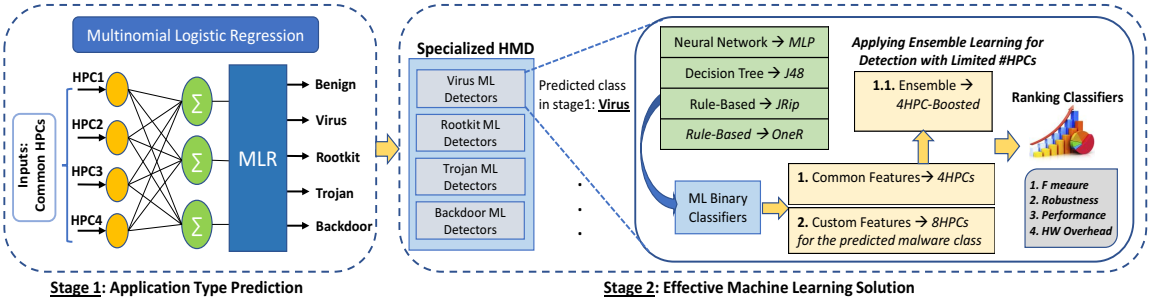

Fig. 3: 2SMaRT overview, the proposed two-stage malware detection approach

Common features. These features include branch instructions (branch inst), cache references (cache ref), branch misses, and node-stores (node-st). Along with the Common features, within each class of malware, we increase the number of HPCs from 4 to 8, referred as Custom features, tuning the ML classifiers individually for the corresponding malware class. To evaluate the effectiveness of the 2SMaRT at the second stage, we implemented all ML classifiers using the 4 Common and the 8 Custom HPCs listed in Table II for each malware class.

Stage 2: Effective Machine Learning Techniques. As observed previously, MLR model alone does not provide a high run-time malware detection rate when using small number of microarchitectural events (equal to the available HPCs). To address the challenge of run-time HMD with high detection performance, we cascade a second stage of detection as specialized HMD which uses various types of ML techniques for per-class analysis. These ML classifiers are chosen based on the predicted class of malware by the MLR. As discussed, the specialized ML classifiers are the ML classifiers trained specifically with the dataset of a specific malware class. As seen from Table I, no unique classifier is the winner in detecting all classes of malware. Thus, employing a specialized classifier at run-time which is the winner for the particular class of malware enhances the malware detection performance. These ML classifiers are shown in the second stage of Fig. 3. The rationale for selecting these machine learning models are: First, they are from different branches of ML; regression, neural network, decision tree, rule-based, and ensemble learning covering a diverse range of learning algorithms which are inclusive to model both linear and non-linear problems. Second, the prediction model produced by these learning algorithms can be a binary classification model which is compatible with the malware detection problem. As a result in 2SMaRT HMD, when an application is running, the optimal set of HPC events are obtained first. This is followed by prediction of application class (benign or one of the classes of malware), based on which the customized ML classifier is utilized for detecting and classifying the malware class.

Boosted-ML using Common HPCs. To address the problem of utilizing only available HPCs i.e., even if it is less than what is deployed in previous solutions, we propose to use ensemble learning on top of the two-stage HMD framework. As depicted in Fig. 3, in the second stage of 2SMaRT, we employ ensemble learning on-top of the specialized traditional ML classifiers to improve the detection rate and performance of specialized classifiers using only 4 HPCs to match the performance of classifiers using custom features (8HPCs). Ensemble learning is a branch of ML which is used to improve the accuracy of general ML classifiers by generating a set of base learners and combining their outputs for final decision [22]. In 2SMaRT, we deploy Adaptive Boosting (AdaBoost) to construct the final classifier and analyze its impact on the performance improvement of malware detection. AdaBoost is one of the most commonly used ensemble learning methods for enhancing the performance of ML algorithms. In AdaBoost, each base classifier is trained on a weighted form of the training set in which the weights depend on the performance of the previous base classifier. Once all the base classifiers are trained, they are combined to produce the final classifier.

IV. Experimental Results and Analysis

In this section, we evaluate 2SMaRT across various performance evaluation metrics including F measure, detection performance, and hardware implementation overhead.

A. Evaluation Metrics

To evaluate the detection rate of 2SMaRT, we consider two important metrics: F measure and detection performance. The F measure (F score) in machine learning is interpreted as a weighted average of the precision (p) and recall (r) which is formulated as $\frac{2\times(p\times r)}{p+r}$. The precision is the proportion of the sum of true positives versus the sum of positive instances and the recall is the proportion of instances that are predicted positive of all the instances that are positive. 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, F measure is also resilient to class imbalance in the dataset which is the case in our experiments.

For a comprehensive evaluation, we further calculate the robustness of classifiers with the aid of Area under the ROC Curve (AUC) metric. The AUC corresponds to the probability of correctly identifying malware and benign applications and robustness is referred to how well the classifier distinguishes between malware and benign classes, for all possible threshold values [8, 13]. Higher AUC indicates better robustness for ML classifiers. Using the calculated AUC values, we define the product of F measure and robustness (F$\times$AUC) as the detection performance metric combining the impact of F measure detection rate and robustness (AUC) of malware detection and classification. Fig. 4 shows the performance evaluation of 2SMaRT using various ML classifiers under a varying number of HPCs and four different malware classes.

B. Evaluation Results

Table III presents the F score results of 2SMaRT across different classes of malware and number of HPCs (16, 8, and 4 HPCs) as well as 4HPCs-boosted malware detector which

represents the ensemble learning-based HMD results. From the results, it can be validated that no unique ML classifier achieves highest F measure across all malware classes. As observed with the reduction in the number of HPCs used for HMD the F score decreases in most of the cases. Plus, OneR classifier is not affected by feature reduction and in majority of cases shows almost constant F measure, since it only employs one HPC feature (branch instructions) to predict the existence of malware. As seen, 2SMaRT with only 4HPCs, but boosted with AdaBoost achieves higher or mostly similar F score to 16/8 HPC-based detectors. By applying AdaBoost on top of the 2SMaRT, we achieve up to 98.9% F score (MLP in Trojan) and almost 92% F score on average across all ML classifiers and malware classes compensating the possible negative impact of feature reduction from 16 HPCs and eliminating the need for multiple runs of application to capture the required HPCs. (d) For heavy-weight classifiers such as MLP, the ensemble learning has adverse effect due to over-fitting.

TABLE III: F measure of 2SMaRT detectors with and without boosting

Class	**Backdoor**				**Rootkit**			
#HPC	16	8	4	4-Boosted	16	8	4	4-Boosted
J48	86.7	79.6	80.4	85.5	94.6	87.7	85.75	91.2
JRip	90.5	90	87.8	87.6	84.1	82.5	80.8	91.5
MLP	94.4	92.4	89.5	90	82.9	82.35	93.8	79.8
OneR	94	94	94	93.8	73.2	73.2	73.18	85.99
Class	**Virus**				**Trojan**			
#HPC	16	8	4	4-Boosted	16	8HPC	4	4-Boosted
J48	94.7	94.5	93.2	96.5	98.8	98	93.2	97.3
JRip	93.6	93.1	93	93.9	98.9	98.2	93.3	94
MLP	68.1	67.6	94.7	95.4	98.6	96.7	98.9	98.9
OneR	97.1	90.2	89	94.8	92.7	92.7	92.7	92.7

Fig. 4 depicts the detection performance results of 2SMaRT. As observed, most classifiers across different malware classes deliver higher performance when they are supplied with 16 and 8 features. By decreasing the number of HPCs, the performance of ML classifiers reduces in majority of the cases showing the potential for applying ensemble learning techniques to boost the performance with fewer available HPCs. For instance, in J48 Backdoor detector by reducing the number of HPCs to 4 and applying AdaBoost, we achieve 40% performance improvement as compared to J48-8HPCs. The 2SMaRT methodology achieves a performance of 74.8% on average when employing 16 HPCs but it drops to 70.9% when employing only 4 HPCs across all malware classes, with robustness being the most impacted parameter. For strong classifiers like Multilayer perceptron (MLP), due to overfitting the performance is degraded in some cases with the increase in the number of HPCs. However, techniques such as dropout can be employed, but at the cost of additional overhead.

To qualitatively validate the efficacy of the proposed 2SMaRT with and without ensemble learning, we present the average performance improvement for all studied malware classes in Table IV. The column '8HPC→4HPC-Boosted' in Table IV denotes the average performance improvement when employing 4HPCs associated with AdaBoost compared to general 8HPCs in the proposed 2SMaRT. It can be observed that the detection performance with ensemble learning-based malware detectors using only the 4 Common HPCs outperforms the performance achieved despite employing 8 or 4 HPCs without ensemble learning. However, a negative improvement is seen in the case of neural networks-based detector (MLP) when AdaBoost is employed. As seen, 3.75%-31.25% performance improvement is achieved with 4HPC-boosted malware detection compared to malware detection

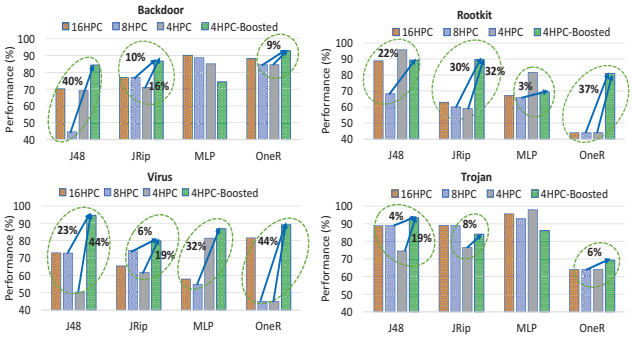

Fig. 4: Malware detection performance (F × AUC) results of 2SMaRT for various ML classifiers across different malware classes

TABLE IV: Average performance improvement of 2SMaRT

ML Classifier	**8HPC→4HPC-Boosted**	**4HPC→4HPC-Boosted**
J48	31.25%	18.2%
JRip	10.1%	18.75%
MLP	3.75%	-6.75%
OneR	24%	24%

employing 8HPCs in which rule-based JRip classifier is achieving the highest improvement. The results clearly confirm the effectiveness of using ensemble techniques in 2SMaRT HMD for performance improvement of ML classifiers with a lower number of HPCs for malware detection. A key point here is that rather than extracting 16 or 8 HPCs which requires multiple runs of the same application and is clearly not a run-time solution, it is more effective to alternatively collect lower number of HPCs equal to available HPCs (four), at lower power and performance cost to the system, and boost the performance of ML classifier with AdaBoost, while facilitating run-time HMD, given the availability of 4HPCs in the system.

2SMaRT vs. Single-Stage HMDs. Here, we present a comparison of detection rate of 2SMaRT against traditional and state-of-the-art single-stage HMDs. Fig. 5-(a) depicts the F measure results of HMD when utilized only first stage (represented as Stage1-MLR) against using the proposed two-stage HMD that accurately detects the type of malware ahead of time (referred as malware_name-2SMaRT). The number of HPCs used for malware detection in Fig. 5-(a) is the 4 Common features. As seen, using only the first stage (MLR) has the lowest F score of 80%. However, in 2SMaRT by using two levels of detection, the malware class is predicted upfront, and the proper ML classifier trained for corresponding malware class is employed which improves the F score by up to 19%.

Furthermore, in Fig. 5-(b) we compare 2SMaRT with a state-of-the-art single-stage HMD proposed in a recent work [2]. We compare 2SMaRT with [2] since it also employs different ML techniques using various number of HPCs to detect the malicious pattern of applications. As seen, the 2SMaRT HMD with only 4HPCs achieves higher detection rate compared to [2] employing 4 and even 8 HPCs, due to the effectiveness of the 2SMaRT two-stage methodology. Given the results, on an average close to 10%, and 9% improvement in detection rate is achieved with 2SMaRT-4HPCs with and without ensemble learning compared to [2] using the same number of HPCs. In addition, interestingly the 2SMaRT with and without AdaBoost technique using only the Common 4HPCs outperforms the malware detectors in [2] with higher number of features (8HPCs) by 9% and 8%, respectively.

Hardware Overhead Analysis. In this section, we evaluate

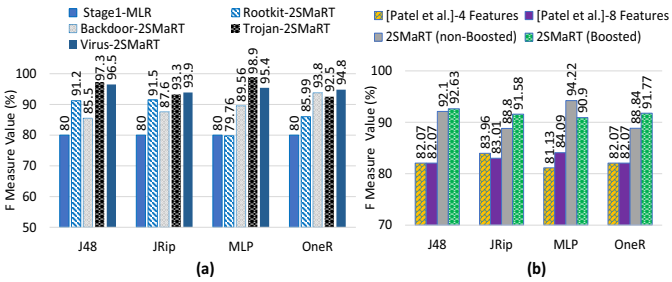

Fig. 5: Comparison of 2SMaRT with state-of-the-art single-stage HMDs

the hardware implementations of 2SMaRT including design area and response time (latency) overhead. We deploy Vivado HLS compiler to develop the HDL implementation of the classifiers on Xilinx Virtex 7 FPGA. This analysis not only helps to have an understanding of the complexity of the proposed solution in terms of number of logic gates (which can be similarly proportional to an ASIC implementation), but also assists in analyzing the design implementation cost in a emerging heterogeneous FPGA+CPU architecture in SoC designs [23, 24]. To evaluate hardware implementation cost, in Table V, we report the results for 2SMaRT classifiers using 8HPCs, 4HPCs, and boosted 4HPCs-based HMD. As detection latency depends on the underlying processor frequency that runs HMD, we present the latency in terms of number of clock-cycles (cycles @10 ns), which includes the latency of first stage and second stage. In order to compare the area overhead of the implemented ML classifiers, we consider the OpenSPARC (FPGA) implementation as reference and calculate the area overhead relative to the core size. The area is the total number of utilized LUTs, FFs, and DSP units in Virtex 7 FPGA. As seen from Table V, the MLP, as expected, results in a significant area and latency overhead, as compared to other learning methods. Using ensemble learning in 2SMaRT with 4HPCs introduces area overhead for some classifiers. However, the introduced overhead is less than 3% compared to the general lightweight classifiers using 8HPCs. This result was expected as the ensemble learning algorithms generate models according to the data sets given and configuration of the algorithm. The rule-based and tree-based classifiers such as OneR and J48 with 4HPC-boosting outperforms non-boosted 8HPC or 4HPC based classifiers.

TABLE V: Hardware implementation results of different detectors

Model Used	**8HPC**		**4HPC**		**4HPC-Boosted**	
Classifier	Latency @10ns	Area (%)	Latency @10ns	Area (%)	Latency @10ns	Area (%)
J48	9	3	3	0.93	67	4.3
JRip	4	2.5	2	0.26	56	5.3
MLP	302	61.1	102	43.2	591	61.7
OneR	1	2.1	1	0.49	70	5.1

V. Conclusion

To realize highly accurate run-time malware detection, this work identified two challenges associated with prior efforts on hardware-assisted malware detection, including limited number of HPCs in modern processors and high variance in ML classifiers detection rate and performance across various classes of malware. In response to these challenges, we proposed 2SMaRT, a complexity-effective two-stage run-time specialized HMD framework. 2SMaRT makes use of the low-level features of microprocessor i.e., HPC events to capture the application behavior, reduces the dimensionality of features by exploiting correlation between different microarchitectural events using correlation analysis and PCA, and analyzes them for malware detection and classification. In the first stage of 2SMaRT, we identify the class of application (benign or a class of malware) using an effective multiclass classification technique. In the second stage, a specialized ML classifiers is used to detect the malware with high performance. Further, an ensemble learning solution is cascaded to address the challenge of employing limited available HPCs. 2SMaRT using only 4HPCs with a lightweight tree-based classifier (J48) boosted by ensemble learning improves malware detection performance on an average by 31.25% compared to the 8HPCs-based HMD.

VI. Acknowledgment

This research was supported in part by DARPA SSITH program under the agreement number HR0011-18-C-0020.

References

[1] G. Jacob and et al., "Behavioral detection of malware: from a survey towards an established taxonomy," *Journal in Computer Virology*, vol. 4, no. 3, pp. 251–266, Aug 2008.
[2] N. Patel and et al., "Analyzing hardware based malware detectors," in *Design Automation Conf.*, 2017.
[3] Q. Chen and R. A. Bridges, "Automated behavioral analysis of malware: A case study of wannacry ransomware," in *ICMLA'17*, 2017.
[4] A. Nazari and et al., "Eddie: Em-based detection of deviations in program execution," in *ISCA'17*. IEEE, 2017, pp. 333–346.
[5] J. Demme and et al., "On the feasibility of online malware detection with performance counters," in *ISCA'13*, 2013.
[6] M. Ozsoy and et al., "Malware-aware processors: A framework for efficient online malware detection," in *HPCA'15*, 2015.
[7] A. Tang and et al., "Unsupervised anomaly-based malware detection using hardware features," in *RAID'14*, 2014.
[8] H. Sayadi and et al., "Ensemble learning for effective run-time hardware-based malware detection: A comprehensive analysis and classification," in *Design Automation Conference (DAC'18)*, 2018.
[9] K. N. Khasawneh and et al., "Ensemble learning for low-level hardware-supported malware detection," in *RAID'15*, 2015.
[10] H. Sayadi and et al., "Machine learning-based approaches for energy-efficiency prediction and scheduling in composite cores architectures," in *ICCD*, 2017.
[11] F. Brasser and et al., "Special session: Advances and throwbacks in hardware-assisted security," in *CASES'18*, Sept 2018, pp. 1–10.
[12] S. Baljit and et al., "On the detection of kernel-level rootkits using hardware performance counters," in *ACM AsiaCCS'17*, 2017.
[13] H. Sayadi and et al., "Comprehensive assessment of run-time hardware-supported malware detection using general and ensemble learning," in *Computing Frontiers (CF'18)*, 2018, pp. 212–215.
[14] H. Sayadi and et al., "Customized machine learning-based hardware-assisted malware detection in embedded devices," in *TrustCom/BigDataSE*, 2018.
[15] B. Sprunt, "The basics of performance-monitoring hardware," *IEEE Micro*, vol. 22, no. 4, pp. 64–71, Jul 2002.
[16] C. Malone and et al., "Are hardware performance counters a cost effective way for integrity checking of programs," in *ACM STC*, 2011.
[17] M. R. Guthaus and et al., "MiBench: A free, commercially representative embedded benchmark suite," in *IISWC'01*, 2001.
[18] M. Helsely, "Lxc: Linux container tools," in *IBM developer works technical library*, 2009.
[19] Intel, "Intel 64 and ia-32 architectures software developer's manual, volume 3b: System programming guide," 2016.
[20] M. Hall and et al., "The WEKA data mining software: An update," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, Nov 2009.
[21] H. M. Makrani and et al., "Energy-aware and machine learning-based resource provisioning of in-memory analytics on cloud," in *SoCC*, 2018.
[22] E. Aghaei and et al., "Ensemble classifier for misuse detection using n-gram feature vectors through operating system call traces," in *Journal of Hybrid Intelligent Systems*, 2017.
[23] H. M. Makrani and et al., "Xppe: Cross-platform performance estimation of hardware accelerators using machine learning," in *ASP-DAC'19*, 2019.
[24] H. M. Kamali and et al., "Lut-lock: A novel lut-based logic obfuscation for fpga-bitstream and asic-hardware protection," in *ISVLSI'18*, 2018.