When Machine Learning Meets Hardware Cybersecurity: Delving into Accurate Zero-Day Malware Detection

Zhangying He[1], Tahereh Miari[2], Hosein Mohammadi Makrani[3], Mehrdad Aliasgari[1], Houman Homayoun[3], and Hossein Sayadi[1]

[1]California State University, Long Beach, CA, USA
[2]California State Polytechnic University, Pomona, CA, USA
[3]University of California, Davis, CA, USA

***Abstract*—Cybersecurity for the past decades has been in the front line of global attention as a critical threat to the information technology infrastructures. According to recent security reports, malicious software (a.k.a. malware) is rising at an alarming rate in numbers as well as harmful purposes to compromise security of computing systems. To address the high complexity and computational overheads of conventional software-based detection techniques, Hardware-Supported Malware Detection (HMD) has proved to be efficient for detecting malware at the processors' microarchitecture level with the aid of Machine Learning (ML) techniques applied on Hardware Performance Counter (HPC) data. Existing ML-based HMDs while accurate in recognizing known signatures of malicious patterns, have not explored detecting unknown (zero-day) malware data at run-time which is a more challenging problem, since its HPC data does not match any known attack applications' signatures in the existing database. In this work, we first present a review of recent ML-based HMDs utilizing built-in HPC registers information. Next, we examine the suitability of various standard ML classifiers for zero-day malware detection and demonstrate that such methods are not capable of detecting unknown malware signatures with high detection rate. Lastly, to address the challenge of run-time zero-day malware detection, we propose an ensemble learning-based technique to enhance the performance of the standard malware detectors despite using a small number of microarchitectural features that are captured at run-time by existing HPCs. The experimental results demonstrate that our proposed approach by applying AdaBoost ensemble learning on Random Forrest classifier as a regular classifier achieves 92% F-measure and 95% TPR with only 2% false positive rate in detecting zero-day malware using only the top 4 microarchitectural features.**

I. Introduction

The last decade has witnessed a never-ending growth in complexity of modern computing systems. This has resulted in emergence of new security vulnerabilities, making these systems accessible targets for increasingly complex cyber attacks [1], [2]. The attackers are actively utilizing emerging vulnerabilities to compromise the security of systems and deploy malicious activities. Malware is the general term for a group of malicious programs developed by cyber-attackers to perform harmful tasks like damaging or disabling computer systems, networks, and mobile devices or leaking sensitive data and personally identifiable information [3], [4]. Attackers typically blend malicious code into seemingly benign and legitimate applications to lure unwary users into downloading malicious programs on the target system. With malware usage continuing to rise, effective malware detection strategies have become more critical as they function as an early warning system to protect the computer systems.

The recent evolution of mobile platforms and Internet-of-Things (IoT) devices has further intensified the impact of malware threats. There exists some important factors influencing the security vulnerability of embedded systems and IoTs including the limited energy and resources available, the low computational capacity, and significant number of computing nodes in the network [5], [6]. Therefore, there is an urgent need to develop intelligent security countermeasures to combat the ever-increasing rise in malicious cyber-attacks and protect the integrity and confidentiality of the authenticated users' information at the computing systems' hardware level.

Traditionally, integrity of data has been protected with various security protocols at the software level with the underlying hardware assumed to be secure. This assumption however is no longer true with an increasing number of attacks reported on the hardware. Conventional software-based malware detection techniques have shown to be inefficient mostly imposing significant complexity and computational overheads on the system. Such detection methods also depend on the static signature analysis of the applications preventing them from detecting unseen attacks at run-time.

To overcome the performance and computational overheads challenges of traditional malware detection techniques, Hardware-Supported Malware Detection (HMD) has emerged by employing low-level microarchitectural features of running applications on the target system [7], [8]. These features are collected through Hardware Performance Counters (HPCs) registers, special-purpose registers designed in modern microprocessors to capture the hardware-oriented events of profiled applications that are left on the underlying processor architecture [4], [9]. Hardware-supported malware detection methods have shown the suitability of standard machine learning (ML) algorithms applied on HPCs information in detecting patterns of malicious applications [7], [10], [4], [11], [12].

Malicious software attacks have continued to evolve in quantity and sophistication during the past decade. Zero-day malware attacks are a form of cyber threat that is released before a defense is in place. They are essentially a type of malware with no history or clear remediation strategy that can be leveraged by threats of different types of malware, such as Viruses, polymorphic worms, Trojans, Rootkits, etc. Hence, the malicious programs can quickly attack the targeted vulnerabilities within the operating system and the applications. Detection of zero-day attacks is a long-standing challenge for

anomaly detection algorithms, especially in security-critical applications. It is known that zero-day malicious software do not have any known antivirus signatures that makes them truly challenging to detect using the traditional off-the-shelf detection mechanisms. In particular, due to the absence of a structured remediation strategy and mitigation plan, such vulnerabilities are exploitable by adversaries. As a result, the existing static signature-based approaches are not a suitable approach to catch zero-day malware applications. In addition, prior works on HMD have not discussed the problem of zero-day malware detection using effective and accurate machine learning techniques.

In this paper, we have addressed the challenge of detecting zero-day malware patterns at run-time using hardware features that has been ignored in prior HMD studies. In particular, our comprehensive examination across different types of malware and machine learning algorithms used for HPC-based malware detection indicates that standard machine learning classifiers (widely used in prior works) fail in recognizing the signature of zero-day (unknown) malware with high detection rate. Our analysis shows a clear performance degradation in standard ML classifiers used for zero-day malware detection. In this work we first present an analysis of existing ML-based malware detection methods utilizing built-in HPC registers information. Next, we comprehensively examine the suitability of various standard ML classifiers for zero-day malware detection and demonstrate that such methods are not capable of detecting unknown malware signatures with high detection rate. Lastly, to address the challenge of run-time zero-day malware detection, we propose an ensemble learning-based technique to enhance the performance of the standard malware detectors despite using a small number of microarchitectural features that are captured at run-time by existing HPCs.

The remainder of this paper is organized as follows. Section II presents an overview of related work and background on the topic of hardware malware detection using machine learning techniques. Section III presents an overview of the proposed methodology. Section IV discusses the evaluation criteria and results analysis. Finally, Section V concludes this study.

II. Background Characterization

In this section, we describe the background on HPCs for security analysis and existing studies on hardware-supported malware detection using machine learning techniques.

A. Hardware Performance Counters for Security Analysis

The complexity of today's computing systems has tremendously increased in the past decades. Hierarchical cache subsystems and processor pipeline, simultaneous multithreading, and out-of-order execution units have a significant impact on the performance of computing systems [13], [14]. Access to the performance monitoring module, an essential feature in modern microprocessors (e.g. Intel, ARM, and AMD), is generally provided in the form of programmable hardware performance counter registers. HPCs are specialized registers designed inside modern microprocessors to monitor and capture different hardware-related events [12], [10]. Due to limited number of physical expensive to implement HPC registers on the processor chip, HPCs are constrained in the number of events that could be counted concurrently [12], [4]. A variety of processor platforms such as Intel, ARM, and AMD include HPCs on their processors. For example, 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 total 8 general purpose counters per core.

TABLE I: HPC features and their descriptions

HPC event	Description
Branch instructions	# branch instructions retired
Branch-misses	# branches mispredicted
Instructions	# instructions retired
bus-cycles	time to make a read/write between the cpu and memory
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	# cache lines into L1 cache from DRAM
L1-icache-load-misses	# instruction misses in L1 instructions cache
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 cache
iTLB-load-misses	# misses in instruction TLB during load operations
Branch-loads	# successful branches

HPCs are able to count a variety of low-level events such as cache memories access and misses, TLB hits and misses, and branch mispredictions for various optimization targets such as performance, energy-efficiency, and security enhancement. In particular, HPCs are programmed to issue an interrupt when a counter overflows or even be set to start the counter from the desired value. Table I reports a subset of deployed low-level features captured by HPC registers from Perf tool under Linux in our experiments and their descriptions.

B. ML for Hardware-Supported Malware Detection

Table II lists a summary of recent ML-based malware detection techniques utilizing HPC features. Demme et al. [7] was the first study to examine the effectiveness of hardware performance counter information for the purpose of accurate malware detection. The authors proposed the idea of using hardware performance counter data to accurately detect malicious behavior patterns using machine learning techniques primarily on mobile operating systems such as Android. The paper ultimately demonstrated successfully the effectiveness of offline machine learning algorithms in identifying malicious software. In addition, it illustrated the suitability of employing HPC information in detecting malware at the Linux OS level such as Linux rootkits and cache side-channel attacks on Intel and ARM processors. It exhibited high detection performance results for Android malware by applying complex ML algorithms, namely Artificial Neural Network (ANN) and K-Nearest Neighbor (KNN).

In a different study, Tang et al. [8] further discussed the feasibility of unsupervised learning that employs low-level HPCs features for detecting return-oriented programming (ROP) and buffer overflow attacks by finding anomalies in hardware performance counter information. For feature selection, this study used the Fisher Score metric to identify the top 7 low level features for malware detection. These reduced features are then used to train unsupervised machine learning methods for detecting deviations in program behavior that occur due to a potential malicious attack. The work further provides a comparison of performance using different sampling frequencies of the HPCs.

The work in [15] used sub-semantic features to detect malware using Logistic Regression (LR) and ANN algorithms. Moreover, they suggested changes in microprocessor pipeline to detect malware in truly real-time nature which increases the

TABLE II: Summary of recent hardware-assisted malware detection techniques and their classification methods

Research	Platform	Classification Model	Threat Type	Microarchitectural Features	Evaluation Metric
[7]	Android, Linux	KNN, NN, DT, RF	Malware	Low-level hardware performance counters in the form of multi-dimensional time series data	FP, ROC, AUC
[8]	Linux	ocSVM	Malware	22 features including LLC, Load & store instructions retired, Branch instructions retired, etc.	F Score, AUC, ROC
[15]	Windows	LR, NN	Malware	Instruction mix features, Memory reference patterns, and Architectural events	ACC, S, C, FP, ROC
[16]	Windows	LR, NN, EL	Backdoor, PWS, Rogue, Trojan, Worm	Instruction mix features, Memory reference patterns, and Architectural events same as [15]	ACC, FP, ROC, AUC
[10]	Linux	SVM, ocSVM, NB, DT	Kernel Rootkits	8 low level events (branch instructions, cache misses, etc.)	Confusion Matrix, ROC
[4]	Linux	BN, J48, JRip, MLP, OneR, RT, SGD, SMO, AB, BG	Malware	(32/16/8/4/2) low-level events (branch instructions, cache misses, etc.)	ACC, AUC, ACC*AUC
[17]	Windows	DT, RF, MLP, KNN, AB, NB	Malware	6 low level features (data cache load & store references, number of CLFLUSH instructions executed, etc.	AUC, F1-score
[12]	Linux	J48, JRip, MLP, OneR, AB	Virus, Trojan, Rootkits, Backdoor	8/4 low-level events (branch instructions, cache misses, etc.)	F Score, AUC, F Score*AUC
[11]	Linux	J48, JRip, LR, KNN, BOFF, FCN	Stealthy Malware (Trojan, Rootkits, Backdoor, Blended)	1 low-level event (branch instructions)	F1-score, AUC, P, R, ACC

Accuracy: ACC, Sensitivity: S, Specificity: C, Precision: P, Recall: R, K Nearest Neighbor: KNN, BayesNet: BN, NaiveBayes: NB, Logistic Regression: LR, AdaBoost: AB, Bagging: BG, Support Vector Machine: SVM, One Class SVM: ocSVM, Neural Netework: NN, Last Level Cache References: LLC, REPTree: RT, Decision Tree: DT, Random Forest: RF, Ensemble Learning: EL, Bag-of-Pattern-Features: BOFF, Fully Convolutional Network: FCN.

overhead and complexity. They explored sub-semantic features in the low-level features space by evaluating three types of features including 1) features based on executed instructions that include frequency of instruction categories, Frequency of opcodes with largest difference, existence of categories, and existence of opcodes 2) features based on memory address patterns consist of frequency of memory address distance histogram and memory address distance histogram mix, and 3) features based on architectural events that include frequency of memory read/writes, taken and immediate branches, and unaligned memory accesses. The HMD study in [16] used the same feature set and applied 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.

Singh et al. [10] is a recent work on HMD that deploys machine learning algorithms applied on synthetic traces of HPC features for detection of kernel-level rootkit attacks. For feature reduction, they process the application traces using the Gain Ratio feature selection technique from the WEKA machine learning toolkit to determine which features are the most prominent for each rootkit. The authors achieve high prediction accuracy in detecting five self-developed synthetic rootkits models. Nevertheless, this work while important only focused on detection of kernel rootkit attacks using a limited set of synthesis datasets.

The research in [4] 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 addition, a recent work in [12] 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 [17] evaluated the suitability of HPCs for HMD. Though the presented experimental results in [17] 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 attack in CNNs for image processing [18]. 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.

The work in [11] focuses on the challenge of detecting embedded malware using hardware features. Embedded malware refers to harmful stealthy cyber-attacks in which the malicious code is hidden within benign applications and remains undetected by traditional malware detection approaches [19]. In HMD methods, when the HPC data is directly fed into a ML 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. To address this challenge, the authors in [11] presents StealthMiner, a specialized time series machine learning approach based on Fully Convolutional Network (FCN) to detect embedded malware at run-time using branch instructions feature, the most prominent HPC feature.

III. Methodology Overview

In this section, we describe the proposed machine learning-based approach for effective run-time zero-day HMD. As depicted in Figure 1 the microarchitectural features are first collected using a performance evaluation tool and then we analyze the features to select the most prominent HPCs addressing issue of run-time detection using a limited number of HPC registers physically available on the modern microprocessors chip. Next, various ML models (regular vs. boosted) will be implemented to detect the existence of zero-day malware.

A. Experimental Configuration

In our experiments, 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 sampling time of 10ms. We executed more than 5000 benign and malware applications for data collection. Benign applications include real-world applications comprising MiBench and SPEC2006, Linux system programs, browsers, and text editors. Malware applications collected from virustotal and virusshare online repositories which comprises nine types of malware including Worm, Virus, Botnet, Ransomware, Spyware, Adware, Trojan, Rootkit, and Backdoor. The HPC information is collected by running applications in an isolated environment referred as Linux Containers (LXC) [20] which unlike common virtual platforms such as VMWare or VirtualBox, provides access to actual hardware performance counters data instead of emulating HPCs.

B. Machine Learning Classifiers

Here we briefly describe the machine learning classifiers tested for known and unknown malware detection in this work. The rationale for choosing these machine learning models is that they are from different branches of ML covering a diverse range of learning algorithms and the prediction model produced by these learning algorithms can be a binary classification model which is compatible with the malware detection problem.

- *Decision Tree (DT)* is a sequential supervised learning model (a.k.a. divide and conquer algorithm), which logically combines a sequence of simple tests where a numerical attribute is compared against a threshold value or against a set of possible values. It is essentially a flow chart like structure where each internal node denotes a test on an attribute with each branch representing an outcome of the test and each leaf holding a class label. Decision Trees tend to overfit data with many features, so feature selection and dimensional reduction of a dataset are essential to avoid overfitting.
- *Random Forest (RF)* is an ensemble machine learning algorithm based on randomized decision trees. It basically builds the tree by splitting on a random subset of features. In each split, the model selects only a small subset of features randomly, so its model tends to have a low bias and a moderate variance. It is a robust machine learning algorithm that can properly handle imbalanced dataset.
- *Gaussian Naïve Bayes (GNB)* is a simple classification technique based on the Bayes theorem. It can handle a high dimensional dataset with a smaller model size. Complex classification problems can also be implemented by using the Gaussian Naive Bayes Classifier. GNB supports continuous data, assuming that the feature sets associated with each class are distributed according to a Gaussian distribution.
- *Stochastic Gradient Descent Classifier (SGD)* is a linear classifier optimized by stochastic gradient descent which is an efficient approach to fitting linear classifiers and regressors under convex loss functions such as (linear) Support Vector Machines (SVM) and Logistic Regression (LR). By default, Scikit Learn implements a stochastic gradient descent learning routine over a linear SVM that supports different loss functions and penalties for classification. It allows minibatch learning that each sample estimates the gradient of the loss at a time. The model is updated along the way with a scheduled decreasing learning rate.
- *Logistic Regression (LR)* is a linear statistical regression-based algorithm used for analyzing a dataset in which there are one or more independent variables that determine an outcome. The goal of this algorithm is to find the best fitting model to describe the relationship between dependent variable (response or outcome variable) and a set of independent (predictor or explanatory) variables. In particular, the logistic regression algorithm models and refines the relationship between features and labels iteratively using error measures and output probabilities of classes.

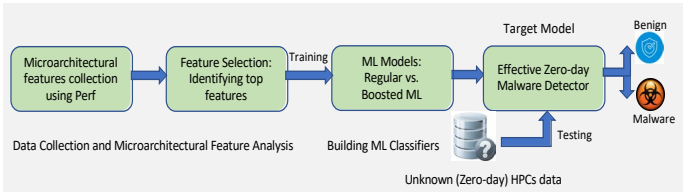

Fig. 1: Overview of the proposed zero-day malware detection framework using microarchitectural features

- *ExtraTree Classifier (ExtraTree)* is a meta estimator to fit multiple randomized sub-trees by randomly sampling data from the original input data without data replicas. It selects the tree split randomly and once the split points are chosen, the two algorithms on the sub-trees determine the best one between all the subset of features. Therefore, ExtraTree adds randomization but still has optimization which has reduced bias and variance than algorithms with a bootstrap replica. Due to the nature of a random sampling of the data, it is a fast algorithm that uses averaging of a subset of trees to improve the predictive accuracy and minimize over-fitting.
- *AdaBoost* or Adaptive Boosting is one of the most commonly used ensemble learning methods for enhancing the performance of ML algorithms. In AdaBoost methodology, each base classifier is trained on a weighted form of the training dataset in which the weights depend on the performance of the previous base ML classifier. Once all the base classifiers are trained, they will be combined to produce the final classifier. Each training instance in the dataset is weighted and the weights are updated based on the overall accuracy of the model and whether an instance was classied correctly or not. Subsequent models are trained and added until a minimum accuracy is achieved or no further improvement is possible. In this work, we applied AdaBoost as a boosting learning technique on all studied general ML classifiers to analyze its impact on the accuracy and performance improvement of HPC-based zero-day malware detection.

C. Hardware Features Analysis

Feature selection is considered as a critical step of developing effective ML-based hardware malware detectors [4]. There exists numerous microarchitectural events with different functionality available to collect from running programs in modern microprocessors. Counting all possible features would result in a high dimensional data which increases computational complexity and induces delay. Moreover, including irrelevant features could reduce the performance of classifiers [21], [12].

We employ the Recursive Feature Elimination (RFE) method using the wrapper function available in Scikit Learn for feature selection. RFE is a feature selection method that fits a model and removes the weakest feature (or features) until the specified number of features is reached. In particular, RFE begins by building a model using our specified DT classifier on the entire dataset, computing the weights of the features. It then gradually eliminates the features with less importance. In the last, the least important features are pruned until the target number of adjusted features (4 HPCs) is met. To this

end, we deploy RFE with a Stratified ten-fold cross-validation technique in our feature selection process. The Stratified cross-validation method shuffles the dataset randomly and splits it into ten groups. Each round holds one group as a test set and uses the rest groups as the training set. It then fits both training and test data to the DT classifier to evaluate the model performance, assigns weights to each feature, and prunes less important features. We run it repeatedly for five times at each experiment with a different number of features. RFE gradually prunes the less important features until we finally rank all captured features and select the top features with satisfying model accuracy. We ultimately choose the top four features of node-loads, dTLB-stores, cycles-ct, and branch-instructions to implement the ML-based malware detectors.

D. Implementation of ML-based Malware Detectors

Existing HMD studies have adopted two main validation methods including cross validation and percentage split to assess the effectiveness of the ML-based malware detectors. The cross validation method splits the dataset into K (1, ..., n) folds and selects one of them as testing dataset while the rest folds are used for training dataset. The number of iteration times is decided by the accuracy increases over successive iterations and stops when the accuracy of the validation set does not increase. Whereas, in the percentage split method, the collected database is split into two parts based on percentage setting allocated to training and the other to testing set. However, the major issue with these validation techniques is that the testing data is split from the large dataset and is part or separated from the same data type used in training dataset. Hence, in majority of scenarios such validation techniques could not imitate the zero-day testing result in real-world applications in which the trained ML classifiers should have never seen the testing dataset.

Our methodology involves a two-stage process. We first select the most prominent HPC features to train the machine learning models with default parameter settings in the first stage as our base learners. In the second stage, we feed these base learners to AdaBoost ensemble learning classifier, run a five-fold model selection process over the training dataset, and evaluated their cross-validation scores to find the best model of each run. We then test the boosted machine learning models across various metrics on the validation and zero-day test dataset. We implemented our ML classifiers with default settings using scikit-learn [22]. Scikit-learn also supports popular boosting algorithms, including AdaBoostClassifier that we used it to enhance the detection rate of the base ML-based malware detectors.

To model the zero-day malware threat type, among all malware types, we hold two types of malware from rootkit and backdoor as the target zero-day test data to imitate the zero-day testing result in real-world applications in which the trained machine learning classifiers should have never seen the testing dataset. The rest seven types of malware are considered for training and validation purposes. We randomly split them into 80% for training and 20% for validation datasets. As demonstrated in Figure 1 and subsection III-V, various regular and boosted ML models are trained and tested using the unknown zero-day dataset to explore the feasibility of the standard and boosted machine learning classifiers in detecting the zero-day malware based on microarchitectural features. Particularly, given the weak performance of standard ML models (as we show in Section IV), we propose to use AdaBoosted Random Forest classifier as the target hardware-supported zero-day malware detector with high detection rate.

IV. Experimental Results and Evaluation

Evaluating the performance of ML classifiers is an important step in implementing effective ML-based countermeasures. For analyzing the detection rate, malicious applications samples are considered as positive instances. Hence, the True Positive Rate (TPR) represents the proportion of correctly identified positive instances or malicious samples. The True Negative Rate (TNR) also evaluates the specificity that measures the proportion of correctly identified bengin or negative samples. In addition, the False Positive Rate (FPR) is the rate of benign files that are wrongly classified as malware.

The F-measure (F1-score) in ML is interpreted as a weighted average of the precision (p) and recall (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. Furthermore, Receiver Operating Characteristic (ROC) is a statistical plot that depicts a binary detection performance while its discrimination threshold setting is changeable. The ROC space is supposed by FPR and TPR as x and y axes, respectively. It determines trade-offs between TP and FP (the benefits and costs analysis). Given that the TPR and FPR are equivalent to sensitivity and (1-specificity) respectively, each prediction result represents one point in the ROC graph in which the point in the upper left corner ([0, 1]) stands for the perfect detection result, indicating 100% sensitivity and 100% specificity. Area Under the Curve (AUC) is another important evaluation metric for checking any ML model's performance at various thresholds settings. It shows how well a classification model is capable of distinguishing between different classes.

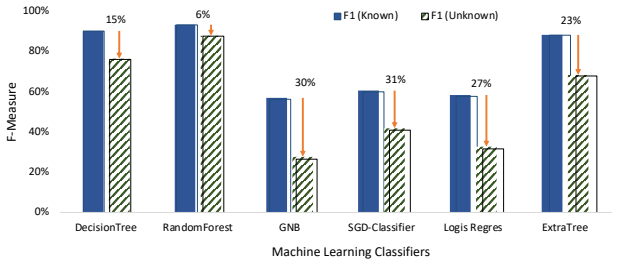

Fig. 2: F-Measure comparison of various ML-based HMD models for detecting known and unknown (zero-day) malware

To further shed light on the challenge of zero-day malware detection using microarchitectural features, we have implemented various ML classifiers (used for HMD with 4 HPC features) considering both known and unknown conditions and the F-measure results are shown in Figure 2. As observed, the performance of standard machine learning models drop (up to 30% in GNB and SGD) when examined by the unknown (zero day) test data such that the trained machine learning classifiers have never seen the testing dataset. This significant performance reduction highlights the necessity of proposing an effective ML-based solution on top of the weak standard

TABLE III: Performance and overhead results of different ML-based detectors for zero-day malware detection

Model	F1-Score	AUC	TPR	FPR	FNR	Latency(ms)
Random Forest	0.88	0.88	0.89	0.02	0.11	0.0160
Decision Tree	0.77	0.79	0.88	0.06	0.12	0.0003
GNB	0.28	0.33	0.25	0.08	0.75	0.0003
SGD	0.42	0.52	0.70	0.22	0.30	0.0013
Logistic Reg.	0.33	0.42	0.55	0.25	0.45	0.0009
ExtraTree	0.68	0.71	0.82	0.08	1.0	0.0004
Boosted-RF	**0.92**	**0.922**	**0.95**	**0.02**	**0.05**	**0.0183**
Boosted-DT	0.8	0.82	0.96	0.06	0.04	0.0004
Boosted-GNB	0.34	0.38	0.39	0.13	0.61	0.0114
Boosted-SGD	0.48	0.65	0.996	0.30	0.004	0.0009
Boosted-LR	0.33	0.58	0.96	0.52	0.04	0.0082
Boosted-ExtraTree	0.76	0.78	0.83	0.05	0.17	0.0005

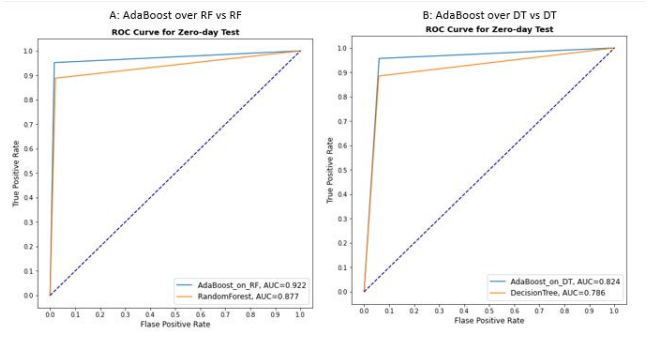

Fig. 3: ROC graphs of the zero-day malware detectors A) Boosted-RF vs. RF, and B) Boosted-DT vs. DT

classifiers to boost up of the detection rate of hardware-supported zero-day malware detection.

Table III reports the performance and latency overhead results of different ML-based detectors (regular and boosted) for zero-day malware detection using 4 HPC features. We essentially ran two rounds of the experiment by applying AdaBoost techniques over the trained most robust machine learning classifiers. We observe that our proposed AdaBoosting technique over Random Forest, which is the strongest classifier among all test models achieves F1-score and AUC of 92% and 92.2% on the unknown dataset, outperforming the regular RF classifier results (0.88 on F1-score and 0.87 on AUC without using the boosting method). The proposed Boosted-RF model also offers 95% TPR with only 2% false positive rate with relatively negligible detection latency per sample overhead.

Figure 3 further illustrates the ROC graphs of zero-day malware detectors for RF and DT models with and without applying AdaBoost method. The solid blue line in the figure shows the ROC curve plot of our proposed method on the zero-day unknown dataset, compared with the solid orange line of the Random Forest before applying AdaBoost. As seen, our ensemble learning-based method improves the ROC curve on the zero-day test dataset from 0.877 to 0.922 in Random Forest classifier, with a 4.5% enhancement highlighting the effectiveness of the proposed method in improving the robustness of the zero-day malware detection.

V. Conclusion

Existing ML-based hardware malware detection methods while offering promising results in detecting known signatures of malicious patterns, fall short in accurate detection of unknown (zero-day) malware at run-time with a limited number HPCs. Since the zero-day malware HPC data does not match any seen attack applications' signatures in the existing database, that makes it a more challenging problem to address. In this work, we first present a review of the progress on ML-based malware detection techniques utilizing built-in HPC registers information. Next, we explore the suitability of various standard ML classifiers for zero-day malware detection and demonstrate that such methods are not able to detect the unknown malware signature with high detection rate. In response, we propose an ensemble learning-based methodology to enhance the performance of the standard ML-based detectors for detecting unknown malware despite using a small number of microarchitectural features that are captured at run-time by existing HPCs.

References

[1] H. Wang and et al., "Mitigating cache-based side-channel attacks through randomization: A comprehensive system and architecture level analysis," in *DATE'20*, 2020.

[2] H. Sayadi and et al., "Recent advancements in microarchitectural security: Review of machine learning countermeasures," in *MWSCAS'20*, 2020, pp. 949–952.

[3] A. Bettany and M. Halsey, "What is malware?" in *Windows Virus and Malware Troubleshooting*. Springer, 2017, pp. 1–8.

[4] H. Sayadi and et al., "Ensemble learning for effective run-time hardware-based malware detection: A comprehensive analysis and classification," in *DAC'18*, 2018, pp. 1–6.

[5] A. Mosenia and N. K. Jha, "A comprehensive study of security of internet-of-things," *IEEE Transactions on Emerging Topics in Computing*, vol. 5, no. 4, pp. 586–602, Oct 2017.

[6] S. M. P. Dinakarrao and et al., "Lightweight node-level malware detection and network-level malware confinement in iot networks," in *DATE'19*, March 2019, pp. 776–781.

[7] J. Demme and et al., "On the feasibility of online malware detection with performance counters," in *ISCA'13*. ACM, 2013, pp. 559–570.

[8] A. Tang and et al., "Unsupervised anomaly-based malware detection using hardware features," in *RAID'14*. Springer, 2014, pp. 109–129.

[9] H. Wang and et al., "Hybrid-shield: Accurate and efficient cross-layer countermeasure for run-time detection and mitigation of cache-based side-channel attacks," in *ICCAD'20*, ser. ICCAD '20, 2020.

[10] B. Singh and et al., "On the detection of kernel-level rootkits using hardware performance counters," in *ASIACCS'17*, 2017, pp. 483–493.

[11] H. Sayadi and et al., "Stealthminer: Specialized time series machine learning for run-time stealthy malware detection based on microarchitectural features," in *GLSVLSI'20*, 2020, p. 175–180.

[12] H. Sayadi and et al., "2smart: A two-stage machine learning-based approach for run-time specialized hardware-assisted malware detection," in *DATE'19*, March 2019, pp. 728–733.

[13] H. M. Makrani and et al., "Xppe: Cross-platform performance estimation of hardware accelerators using machine learning," in *ASP-DAC'19*, 2019.

[14] H. M. Makrani and et al., "Pyramid: Machine learning framework to estimate the optimal timing and resource usage of a high-level synthesis design," in *FPL'19*, 2019, pp. 397–403.

[15] M. Ozsoy and et al., "Malware-aware processors: A framework for efficient online malware detection," in *HPCA'15*, 2015, pp. 651–661.

[16] K. N. Khasawneh and et al., "Ensemble learning for low-level hardware-supported malware detection," in *RAID'15*, 2015, pp. 3–25.

[17] B. Zhou and et al., "Hardware performance counters can detect malware: Myth or fact?" in *ASIACCS'18*, 2018, pp. 457–468.

[18] I. J. Goodfellow and et al., "Explaining and harnessing adversarial examples," in *arXiv:1412.6572*, 2015.

[19] S. J. Stolfo and et al., "Towards stealthy malware detection," in *Malware Detection*. Boston, MA: Springer US, 2007, pp. 231–249.

[20] M. Helsely, "Lxc: Linux container tools," in *IBM developer works technical library*, 2009.

[21] H. Liu and et al., *Feature selection for knowledge discovery and data mining*. Springer Science & Business Media, 2012, vol. 454.

[22] F. Pedregosa and et al., "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.