Ensemble Learning for Effective Run-Time Hardware-Based Malware Detection: A Comprehensive Analysis and Classification

Hossein Sayadi, Nisarg Patel, Sai Manoj P D, Avesta Sasan, Setareh Rafatirad, Houman Homayoun
George Mason University, Fairfax, VA, USA 22030
{hsayadi, npatel33, spudukot, asasan, srafatir, hhomayou}@gmu.edu

ABSTRACT

Malware detection at the hardware level has emerged recently as a promising solution to improve the security of computing systems. Hardware-based malware detectors take advantage of Machine Learning (ML) classifiers to detect pattern of malicious applications at run-time. These ML classifiers are trained using low-level features such as processor Hardware Performance Counters (HPCs) data which are captured at run-time to appropriately represent the application behaviour. Recent studies show the potential of standard ML-based classifiers for detecting malware using analysis of large number of microarchitectural events, more than the very limited number of HPC registers available in today's microprocessors which varies from 2 to 8. This results in executing the application more than once to collect the required data, which in turn makes the solution less practical for effective run-time malware detection. Our results show a clear trade-off between the performance of standard ML classifiers and the number and diversity of HPCs available in modern microprocessors. This paper proposes a machine learning-based solution to break this trade-off to realize effective run-time detection of malware. We propose ensemble learning techniques to improve the performance of the hardware-based malware detectors despite using a very small number of microarchitectural events that are captured at run-time by existing HPCs, eliminating the need to run an application several times. For this purpose, eight robust machine learning models and two well-known ensemble learning classifiers applied on all studied ML models (sixteen in total) are implemented for malware detection and precisely compared and characterized in terms of detection accuracy, robustness, performance (accuracy×robustness), and hardware overheads. The experimental results show that the proposed ensemble learning-based malware detection with just 2 HPCs using ensemble technique outperforms standard classifiers with 8 HPCs by up to 17%. In addition, it can match the robustness and performance of standard ML-based detectors with 16 HPCs while using only 4 HPCs allowing effective run-time detection of malware.

KEYWORDS

Malware Detection, Hardware Performance Counters, Ensemble Learning

1. INTRODUCTION

Malware is a piece of code designed to perform various malicious activities, such as destroying the data, stealing information, running destructive or intrusive programs on devices to perform Denial-of-Service (DoS) attack, and gaining root access without the consent of user. According to a 2017 McAfee threats report [12], 57.6 million new malware samples have been recorded in the third quarter of 2017, an all-time highest number with an increase of 10% from the second quarter. Furthermore, the overall counts of new malware samples grew by 27% in 2017 to 781 million samples. The recent proliferation of computing devices in mobile and Internet-of-Things domains further exacerbates the malware attacks and calls for effective malware detection techniques.

Malware detection can be simplified as a binary classification problem regardless of what detection method is being used. It is basically envisioned as distinguishing whether the running application has malicious intent or not. Traditional malware detection approaches such as signature-based detection and semantics-based anomaly detections are considered as software-based solutions and incur significant computational overheads [10]. Recent studies have demonstrated that malware behavior can be differentiated from benign applications by classifying anomalies in the low-level feature spaces such as microarchitectural events collected by Hardware Performance Counter (HPC) registers [3,4,5,11,13,15,16,24]. HPCs are CPU hardware registers that count hardware events such as instructions executed, cache-misses suffered, or branches mispredicted. Performance counters data have been extensively used to predict the power, performance, and energy efficiency of computing systems [14,20,22], and recently drew attentions to be used for detecting the malicious pattern of running applications to improve the security of systems. Thus, malware detection using HPCs microarchitectural events has emerged as a promising alternative to traditional malware detection methods [3,4,11,13,24]. As learning the underlying patterns of these microarchitectural events can aid in detecting malware, machine learning (ML) techniques are widely deployed for malware detection. The HPC microarchitectural features are used to train ML-based classifiers. In addition, such ML-based malware detection methods 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 (few clock cycles) [4].

Recently, there has been a number of work on hardware-based malware detection using HPCs information [3,4,11,13,14,24]. However, these works performed a limited study on malware classification accounting for the availability of a large number (e.g. 16 or 32) and diverse type of HPCs. While, modern processors in the high-performance domain have a small number of HPCs (2 to 8), due to several reasons including the design complexity and cost of concurrent monitoring of microarchitectural events [17,21,23]. Due to deep pipelines, complex prefetchers, branch predictors, modern cache design etc., HPCs implementation becomes a great challenge in terms of counting multiple events and maintaining counter accuracy at the same time under speculative execution [17]. Better accuracy requires better and more complex hardware design hence increasing the number of counters with limited accuracy doesn't appear to be a good trade-off. Even modern Intel Xeon architectures houses only 4-6 performance counters, compare to 2 in Pentium 4 and server class Intel

DAC '18, June 24–29, 2018, San Francisco, CA, USA

ACM ISBN 978-1-4503-5700-5/18/06…$15.00

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.

Therefore, collecting a variety of microarchitectural events, more than the number of available HPCs, to achieve high accuracy using the general ML models presented in prior work, requires running 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. In addition, previous studies, mostly focus on specific learning classifiers and limited types of malware [3,11,13]. A quantitative comparison of these studies indicates that there is no unique classifier that delivers the best results across various metrics including performance (accuracy and robustness) and area overhead as well as detection delay and various classes of malware.

As the performance of malware detection depends on the type of ML classifier applied and the number and type of HPC events used, in this work, we first illustrate the impact of ML classifier type on malware detection accuracy and performance and the effect of number of HPC events for malware detection. To achieve a high accuracy across all studied general ML classifiers, of more than 80%, at least 16 hardware performance counters are required, which as discussed is not available in modern processors, even in the high-performance domain, making run-time detection of malware impractical using these methods. Therefore, a key challenge in making the hardware-based malware detection a practical run-time solution is how to use a limited number of HPCs available in a microprocessor (for instance 2 or 4) and match the accuracy and performance of malware detection with the ones that can be achieved by a larger number of HPC events (for instance 16 or 32). In this work, we address this challenge by proposing ensemble learning techniques to improve the accuracy and performance of the hardware-based malware detectors and break the trade-off between accuracy/performance with respect to the number of HPCs. We explore the effectiveness of ensemble learning models in 1) reducing the number of required performance counters for implementing effective ML classifiers for run-time malware detection and 2) improving the performance of weak but low-cost classifiers in malware detection with a small number of HPCs.

The remainder of this paper is organized as follows. The background of ensemble learning is briefly described in section 2. The proposed hardware-based malware detection framework and experimental setup details are discussed in Section 3. Section 4 presents the experimental results and provides a comprehensive analysis of different malware detectors across various metrics. Then, we present the state-of-the-art works on HMD in section 5. Finally, Section 6 presents the conclusion of this study.

2. ENSEMBLE LEARNING

Ensemble learning is a branch of machine learning which is used to improve the accuracy and performance of general ML classifiers by

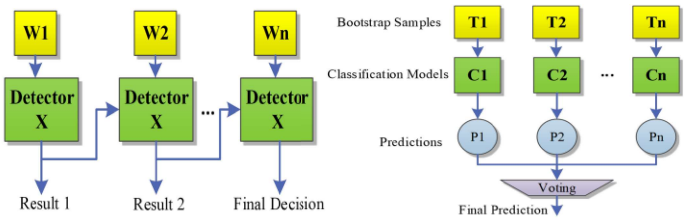

Figure 1: Ensemble learning block diagrams a) AdaBoost, b) Bagging

generating a set of base learners and combining their outputs for final decision. It fully exploits complementary information of different classifiers to improve the decision accuracy and performance. The ensemble learning, and joint decision procedure are widely used to devise learning methods to achieve more accurate predictions and stronger generalization performance. In this work, we deploy and analyze the effectiveness of two ensemble learning methods for efficient malware detection even with less number of HPCs. These ensemble methods are briefly described in below:

Boosting is one of the most commonly used ensemble learning methods for enhancing the performance of ML algorithms. Adaptive Boosting, or in short AdaBoost [18], is the first proposed implementation of this type of ensemble learners. Figure 1-a illustrates the AdaBoost methodology. As shown, 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 classified 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 hardware-based malware detection.

Bagging, or Bootstrap Aggregation [19] is an ensemble learning model that is used for classification and regression problems. It is a statistical prediction technique where a statistical value like a mean is estimated from multiple random samples of training data which are drawn with replacement and used to train different ML models. Each model is then exploited to make a prediction and the results are averaged to give a more robust and generalized prediction. Figure 1-b illustrates the overview of bagging model. Bagging is a technique that is best used with models with low bias and high variance, in which the predictions of base learners are highly dependent on the data from which they were trained. Therefore, it is most suited for our purpose, given the wide variation in ML classifier performance as we will show later in this work. The most used algorithm for bagging that fits the requirement of high variance are decision trees [19].

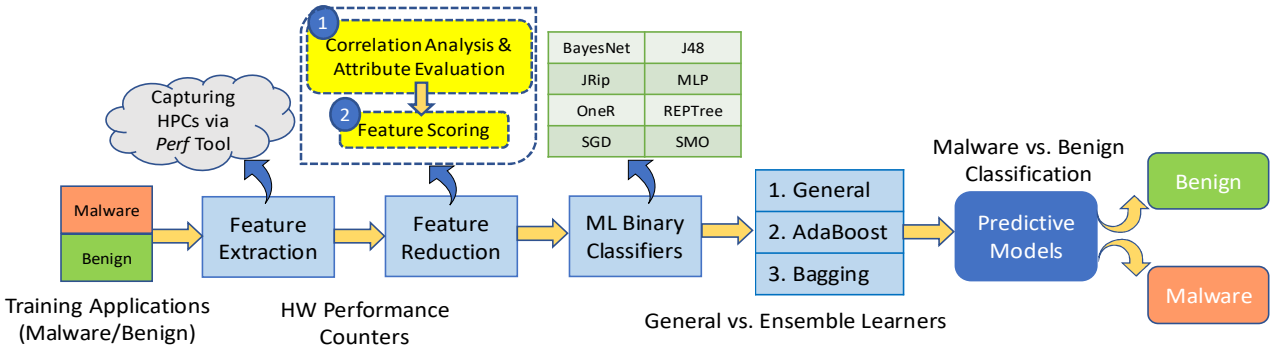

Figure 2: The overview of proposed hardware-based malware detection approach

3. MALWARE DETECTION FRAMEWORK

In this section, we present the details of our proposed run-time hardware-based malware detection approach.

3.1 Experimental Setup and Data Collection

This section provides the details of the experimental setup and data collection procedure. We run all applications on an Intel Xeon X5550 machine running Ubuntu 14.04 with Linux 4.4 Kernel and collect various HPCs data. This processor is based on Intel's Nehalem design, providing four performance counter registers. In order to extract the HPC information, we use *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 have executed more than 100 benign and malware applications for HPC data collection. Benign applications include MiBench benchmark suite [6], Linux system programs, browsers, text editors, and word processor. For malware applications, Linux malware is collected from virustotal.com [1]. Malware applications include Linux ELFs, python scripts, perl scripts, and bash scripts, which are created to perform malicious activities. After collecting microarchitectural events using *Perf*, we use WEKA tool [7] for evaluating the accuracy and performance of various machine learning classifiers.

Figure 2 depicts the overview of the proposed hardware-based malware detection approach and training the ML classifiers for predicting the malicious behavior of applications. It is primarily composed of various stages including feature extraction, feature reduction, and ML classifiers (general and ensemble) implementation for malware detection which will be discussed in more details in sections 3.2 and 3.3. HPC information is collected by executing all applications in Linux Containers (LXC) which is an isolated environment [8]. LXC is an operating system level virtualization method that shares the same kernel with the host operating system. In this work, 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. We extracted 44 CPU events available under *Perf* tool. Since Intel Xeon has only 4 counter registers available [9], we can only measure 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 due to the previous run, the container is destroyed after each run.

3.2 Feature Selection

As mentioned earlier, detecting malware using machine learning models requires representing programs at low microarchitectural level. This process produces very high dimensional dataset. Running ML algorithms with large HPCs would be complex and slow. Besides, incorporating irrelevant features would result in lower accuracy for the classifier [24]. Therefore, instead of accounting for all captured features, irrelevant data is identified and removed using a feature reduction algorithm and a subset of HPCs is selected that includes the most important features for classification. The features are supplied to each learning algorithm and the learning algorithm attempts to find a correlation between the feature values and the application behavior to predict the malware or benign type.

Table 1: Hardware performance counters in order of importance

Hardware Performance Counters		
1- branch_instructions	2- branch_loads	3- iTLB_load_misses
4- dTLB_load_misses	5- dTLB_store_misses	6- L1_dcache_stores
7- cache_misses	8- node_loads	9- dTLB_stores
10- iTLB_loads	11-L1_icache_load_misses	12- branch_load_misses
13- branch_misses	14- LLC_store_misses	15- node_stores
16-L1_dcache_load-misses		

As discussed, the key aspect of building an accurate detector is finding the right features to characterize the input data. We started from 44 performance counters. As shown in Figure 2, after feature extraction, the feature reduction process reduces the number of low-level features. We first use Correlation Attribute Evaluation on our training set under WEKA to monitor the most vital microarchitectural parameters to capture application characteristics. Next, the features are scored based on their importance and relevance to the target variable through the feature scoring process. By applying the feature reduction method, the sixteen most related hardware performance counters are determined and numbered in order of importance for malware detection. These HPCs are listed in Table 1. They are included in our prediction model as input parameters. The selected features include HPCs representing pipeline front-end, pipeline back-end, cache subsystem, and main memory behaviors influential in the performance of standard applications.

3.3 Training & Testing the Malware Detectors

In this section, we describe the details of training and testing the ML classifiers for malware detection. Training involves profiling the incoming application with *Perf* tool under Linux and extracting low-level feature values for each training program, reducing the extracted features to the most vital performance counters, and developing a learning model from the training data. It is important to note that the input variables in our classifiers are the HPCs extracted every 10ms interval from the running applications, and the output variable is the class (malware vs. benign) of an application. For each ML classifier, we construct the general and ensemble models (AdaBoost and Bagging) to detect the malware. In order to validate each of the utilized ML classifiers, a standard 70%-30% dataset split for training and testing is followed. To ensure a non-biased splitting, 70% benign- 70% malware application for training (known applications) and 30% benign-30% malware applications for testing (unknown applications) are used.

4. EXPERIMENTAL RESULTS

In this section, we present the evaluation results for different machine learning classifiers. We thoroughly compare these learning techniques

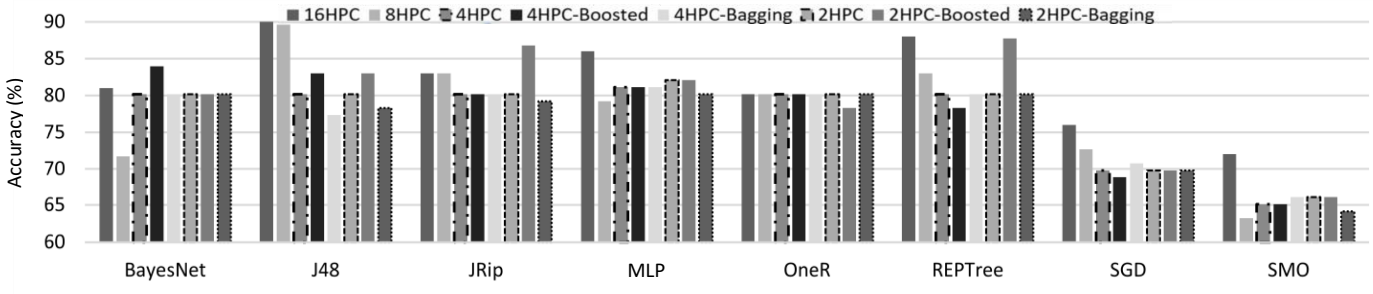

Figure 3: Accuracy results for various ML classifiers with varying number of HPCs

in terms of the prediction accuracy, robustness, performance, and the hardware implementation costs.

4.1 Detection Accuracy

To evaluate the detection accuracy of our malware classifiers, we calculate the percentage value of samples that are correctly classified. Figure 3 shows a comprehensive accuracy comparison of various ML classifiers (general and ensemble) used for malware detection. We have implemented 8 general ML classifiers and two ensemble learning techniques and calculated their accuracy in classifying malware and benign applications. The accuracy of malware detection with different number of hardware performance counters (16, 8, 4 and 2) are reported. Before feature reduction (16 HPCs), most ML classifiers perform well, mostly providing above 80% accuracy. Feature reduction has noticeable impact on the accuracy of several classifiers. However, OneR classifiers perform well even after feature reduction. The reason that OneR classifier is not affected by feature reduction and shows almost constant accuracy results is that it only selects one performance counter (branch_instructions) to predict the malware behavior.

As can be seen in Figure 3, in some classifiers like BayesNet, JRip, OneR, REPTree, and SMO by reducing the number of hardware performance counters to 2 or 4 and applying ensemble learning techniques, a higher or similar accuracy level to 8/16 HPC models is achieved. This interesting observation confirms the effectiveness of using ensemble learning to boost the accuracy of classifiers. For instance, as shown, REPTree achieves close to 88% accuracy with 16 HPCs. However, we observe that reducing the number of vital performance counters to 2 and applying AdaBoost ensemble technique result in achieving almost the same 88% accuracy, as with 16 HPCs.

4.2 Classification Robustness

To evaluate the accuracy and robustness of ML classifiers in detecting malware, Receiver Operating Characteristics (ROC) graphs are used. The ROC curve is produced by plotting the fraction of true positives versus the fraction of false positives for a binary classifier as the threshold changes. The best possible classifier would thus yield a point in the upper left corner or coordinate (0,1) of the ROC space, representing 0% false positives and 100% true positives.

We use the Area Under the Curve (AUC) measure for ROC curve in the evaluation process to examine the robustness of each ML classifier. The area under the ROC curve corresponds to the probability of correctly identifying which application is "malware" and which is "benign". In other words, the AUC measure is more related to the robustness of the classifier. In this work, robustness term is referred to how well the classifier distinguishes between binary malware and benign classes, for all possible threshold values. The AUC value of the best possible classifier is equal to 1, which means that we can find a discrimination threshold under which the classifier obtains 0% false positives and 100% true positives. Table 2 presents the list of the area under the ROC graphs values for each ML general and ensemble classifier with varying number of HPCs. It primarily presents the values for the ROC curves resulted from all comparisons between the general and ensemble-based detectors. A higher AUC value means that the ROC graph is closer to the optimal threshold and the classifier is performing better in terms of classification of malware and benign applications. Area under the curve analysis provides valuable insights to select possibly optimal ML classifiers suitable for malware detection and to discard the suboptimal detectors.

Table 2: AUC values for various general and ensemble detectors

Classifier	16HPC	8HPC	4HPC	4HPC-Boosted	4HPC-Bagging	2HPC	2HPC-Boosted	2HPC-Bagging
BayesNet	0.92	0.92	0.92	0.92	0.94	0.92	0.87	0.93
J48	0.88	0.88	0.81	0.94	0.85	0.81	0.92	0.82
Jrip	0.86	0.86	0.81	0.88	0.93	0.81	0.93	0.88
MLP	0.9	0.9	0.89	0.92	0.86	0.9	0.93	0.87
OneR	0.81	0.81	0.81	0.9	0.87	0.81	0.9	0.87
REPTree	0.85	0.85	0.81	0.85	0.88	0.81	0.92	0.91
SGD	0.74	0.74	0.72	0.89	0.74	0.71	0.71	0.71
SMO	0.65	0.65	0.65	0.88	0.85	0.68	0.89	0.83

Figure 4 depicts the ROC curves for two different ensemble learning models and different number of performance counters. Due to space limitation, here we present the ROC graphs for selected ML classifiers and show the impact of ensemble learning techniques on AUC robustness. In Figure 4-a, the ROC graphs for 4 ML classifiers improved by Bagging ensemble learner are shown which were developed with 4 performance counters. As can be seen in this figure as well as Table 2, the BayesNet and JRip classifiers have maximum AUC of 0.937 and 0.932, respectively, delivering best robustness with only 4 performance counters. Figure 4-b represents the AdaBoost technique effectiveness on two different detectors when reducing the number of HPCs from 8 to 2. As shown, for each classifier boosting model significantly improve the AUC of ROC curve making the ML classifier more effective in terms of classification robustness.

4.3 Performance of Malware Detection

In order to evaluate and compare the performance of malware detectors, we consider the product of accuracy and area under the ROC graph (ACC*AUC) as a performance metric. This metric combines the impact of accuracy and robustness in malware classification and concurrently accounts for both measures. We accounted for performance as a final comparison metric across various ML classifiers since it is a more comprehensive measure by considering both impacts of the detection accuracy and AUC values. Figure 5 illustrates the ACC*AUC results of various ML classifiers under a varying number of hardware performance counters.

As can be seen in the results, most of the classifiers such as JRip, J48, Multi-Layer Perceptron (MLP), and SMO deliver higher performance when they are supplied with 16 and 8 performance counters and by decreasing the number of performance counters, the performance of general ML classifiers decreases showing the potential for applying ensemble learning techniques to boost the accuracy and performance with fewer performance counters. For instance, in SMO classifier by reducing the number of performance counters to 4 and 2 and applying AdaBoost ensemble technique, we achieve 16% and 17% performance improvement, respectively. In REPtree classifier, 2HPC-

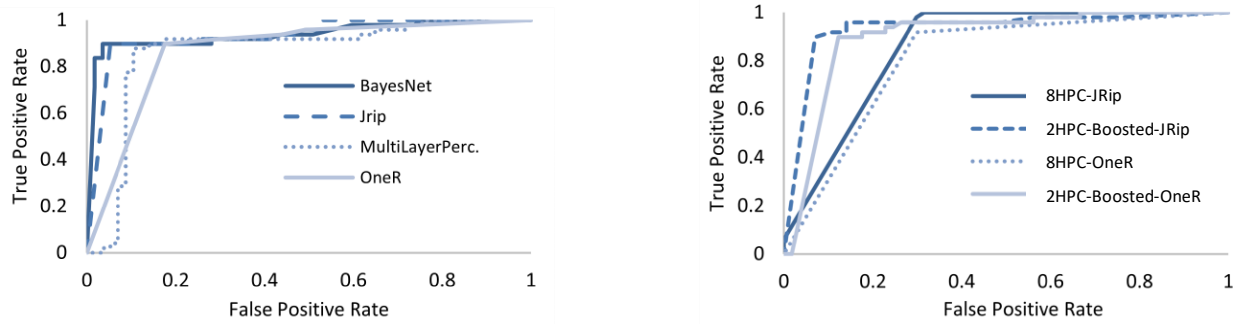

Figure 4: ROC graphs for a) 4HPC-Bagging, b) 8HPC vs. 2HPC-Boosted

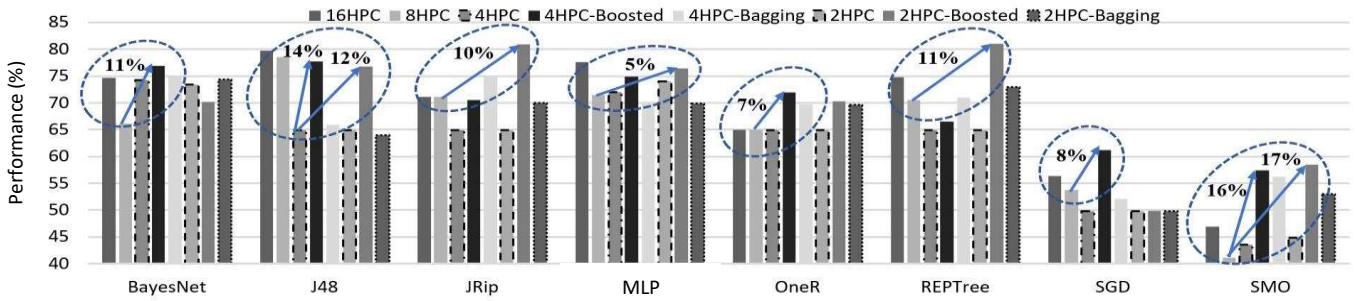

Figure 5: Performance results (ACC*AUC) for various ML classifiers with varying number of HPCs

Boosted detector is achieving 11% improvement in ACC*AUC measure as compared to the general ML classifier with 8 performance counters. JRip classifier achieves 10% performance improvement by applying boosting method and 7% improvement with Bagging technique with the use of only 4 performance counters compared to using 8 HPCs.

The results clearly confirm the effectiveness of using ensemble techniques for performance improvement of ML classifiers with a lower number of HPCs for malware detection. The key point here is that rather than extracting 16 or 8 hardware performance counter which definitely impose significant implementation cost overhead to the systems in terms of resource utilization and power consumption, it is more effective to alternatively collect lower number of HPCs (four or two), depending on the classifier type, and boost the performance of the ML classifier with one of the ensemble learning approaches to improve the accuracy as well as the robustness of malware detectors.

4.4 Hardware Implementation

The software implementation of ML classifiers for malware detection is slow in the range of tens of milliseconds which is an order of magnitude higher than the latency needed to capture the malware at run-time [4]. Therefore, in this paper, we develop a hardware implementation of the general and ensemble learning detectors. We use Vivado HLS compiler to develop the HDL implementation of the classifiers and deploy on Xilinx Virtex 7 FPGA. FPGA is a target in our study, as few modern microprocessors have on-chip FPGAs available for programmable logic implementation. Such arrangement makes it feasible to implement reprogrammable low-level malware detection logic (ML model) which can detect malware by reading the CPU hardware performance counters through shared memory bus. When it comes to choosing the ML classifiers for hardware implementation, the accuracy of an algorithm is not the only parameter in decision-making. Design area and response time (latency) overhead of ML classifiers also plays a key role in selecting the cost-efficient hardware solution. While complex algorithms such as Neural Networks can deliver high accuracy, they will also add significant overhead in terms of hardware implementation cost. Also given their complexity, they can be slow in detecting malware.

In order to compare hardware implementation costs, in Table 3, we report the results for general classifiers using 8 HPCs and boosting ensemble method (AdaBoost) applied on each classifier using 4 and 2 most important HPCs. Latency unit is represented in terms of the number of clock cycles (cycles @10 ns) required to classify input vector. In order to compare the area overhead of the implemented hardware-based ML classifiers, we consider the OpenSPARC (FPGA) implementation as reference and calculate the area overhead relative the core size. The area is the total number of utilized LUTs, FFs, and DSP units inside Virtex 7 FPGA. As can be seen from Table 3, the Multi-Layer Perceptron algorithm, as expected, results in a significant area and latency overhead, as compared to other learning methods.

Table 3: Hardware implementation results

	8HPC-General		4HPC-Boosted		2HPC-Boosted	
Classifier	Latency @10ns	Area (%)	Latency @10ns	Area (%)	Latency @10ns	Area (%)
BayesNet	14	11.5	56	13.6	32	10.9
J48	9	3	67	4.3	35	4.1
SGD	34	4.3	87	6.3	51	5.1
JRip	4	2.5	56	5.3	37	8.2
MultiLperc.	302	61.1	591	61.7	201	42.2
OneR	1	2.1	70	5.1	38	5
REPTree	39	2.9	60	3.9	30	3.7
SMO	34	4.3	87	6.3	51	5.1

The ensemble learning introduces area overhead for some classifiers. However, the introduced overhead is less than 3% compared to the general ML classifiers using 8 HPCs for malware detection. In addition, in some other classifiers we observe that by using ensemble learning with a lower number of performance counters, the area overhead is significantly reduced, compared to the general classifiers using 8 HPCs. For instance, as reported in the previous section, the Boosted-MLP with 2 HPCs gains 5% performance improvement, while as shown in Table 3, it interestingly shows close to 19% area reduction in 2 HPC case and only 0.6% increase in 4 HPCs which is negligible, as compared to the general detectors with 8 HPCs. Ensemble learning algorithms generate models according to the data sets given and configuration of the algorithm. We observe that some algorithms do not see reduction in area from 4 HPCs to 2 HPCs. This is because such algorithms generate same number and equally complex models due to their nature. For instance, JRip-Boosted generates 10 models with 4 HPCs and 10 models with 2 HPCs, hence it is not guaranteed that the area of the 2 HPCs will be less than 4 HPCs. Because JRIP is a rule-based learning algorithm and the area highly depends on how many rules are generated for each model and the 2 HPCs case may have more rules per model.

To the best of our knowledge, there has been no prior work available that discusses the area costs for implementing ML classifiers as a function of HPCs. It can be argued that the number of HPCs can be increased during design time. However, there are several studies available such as [17, 21, 23] that discuss and justify the limited number of HPCs due to complex microarchitecture of modern microprocessors. Because of deeper pipelines, modern complex cache design and etc., implementing the hardware performance counter registers becomes a challenge issue in terms of counting multiple microarchitectural low-level features and at the same time maintaining the accuracy, while achieving higher accuracy requires better and more complex hardware design. As a result, increasing the number of HPCs with limited accuracy doesn't appear to be a good trade-off. Compare to that, ensemble learning algorithm such as AdaBoost can be easily implemented on the programmable logic present in modern heterogeneous microprocessors. Clearly, the results show some trade-offs between the accuracy, latency, and area overhead. Therefore, it is

important to compare classifiers by taking all of these parameters into consideration.

5. RELATED WORK

In this section, we discuss the latest efforts on hardware-based malware detection. The work in [3] was the first study that proposed to use hardware performance counters data for malware detection and demonstrated the effectiveness of offline machine learning algorithms in malware classification. They showed high detection accuracy results for Android malware by applying complex ML algorithms, namely Artificial Neural Network (ANN) and K-Nearest Neighbor (KNN). Although they have discussed implementing classifiers on hardware, they did not present any hardware overhead analysis results. The hardware implementation overhead, particularly area and latency are important as they decide which ML classifier responds in real-time and performs most cost-efficient.

The researchers in [5] and [15] discussed the feasibility of unsupervised learning method on low-level features to detect Return-oriented programming (ROP) and buffer overflow attacks by finding an anomaly in hardware performance counters information. Although unsupervised algorithms can be more effective in detecting new malware and attacker evolution, they are complex in nature requiring more sophisticated analysis, resulting in complex hardware implementations. Also, their software implementation is not an effective solution to detect malware at run-time, due to large latency to compute the complex algorithms. In a different study in [13], the authors used sub-semantic features rather than performance counters to detect malware. Moreover, they suggested changes in microprocessor pipeline to detect malware in truly real-time nature. They discussed estimated latency and area utilization of Logistic and ANN algorithm implementation for their architecture. However, our work is different as it does not require any change in the processor pipeline. The work in [2] collected hardware performance counters to construct support vector machine (SVM) detectors to identify malicious programs in real-time. However, they haven't discussed HW implementation and analysis of those classifiers.

The work in [11] used logistic regression to classify malware into different types and trained a specialized classifier for detecting each type. In their ensemble learning implementation, they limited their experiments on just combining classifiers. In addition, they have ignored to account for the impact of reducing the number of HPCs on the performance of detectors. Our work is different, as we thoroughly study various ML classifiers from different type to investigate the effectiveness of each model in malware detection. Moreover, we explore the effectiveness of different ensemble learning techniques to boost the accuracy and performance of the malware detectors. The prior works, mostly focus on a particular learning classifier and limited type of malware. A quantitative comparison of these works shows that there is no unique classifier that delivers the best results across various metrics including performance (accuracy and robustness), area overhead as well as detection delay and various type of malware. Given that, in this work we thoroughly examined various general and ensemble learning techniques in terms of accuracy, robustness, performance, and hardware implementation costs such as area and latency.

6. CONCLUSION

Hardware-based detectors rely on machine learning classifiers and use HPCs information at run-time. A comparison of recent works on ML-based malware detectors shows that there is no unique general classifier that delivers the best results in terms of performance (accuracy and robustness), area overhead as well as detection delay across various types of malwares. In addition, prior studies mostly relied on a large number of HPCs to gain high accuracy making them less practical for run-time detection using very limited number of HPCs available in modern processors. In this paper, we showed a clear trade-off between the type and count of HPCs and malware classifier performance. To achieve a high accuracy and performance of more than 80% across all studied general ML classifiers, at least 16 HPCs are required, far beyond 2-8 HPCs available in modern architectures. In response to this challenge, this paper proposed using ensemble learning classifiers to boost the performance of general ML classifiers such that by only using 2-4 HPCs they can match the performance of 8-16 HPCs. The proposed ensemble classifiers are applied on 8 general ML classifiers and the results are comprehensively evaluated in terms of accuracy, robustness, performance, and hardware design overhead. The experimental results show that in all studied cases, boosting techniques improves the performance of malware detection classification by up to 17% while using a significantly lower number of performance counters. Given the implementation cost of on-chip HPCs and their limited availability and accuracy, the results of this research will help in making an architectural decision on the number and types of HPCs needed to implement in future architectures, to most effectively improve the performance of ML classifiers for detecting the malicious software.

ACKNOWLEDGMENT

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

REFERENCES

[1] Virustotal intelligence service. http://www.virustotal. com/ intelligence/. Accessed: December 2017.
[2] Bahador et al., "Hpcmalhunter: Behavioral malware detection using hardware performance counters and singular value decomposition", In IEEE ICCKE, 2014.
[3] J. Demme et al., "On the feasibility of online malware detection with performance counters", In ACM SIGARCH Computer Architecture News, Vol. 41, 2013.
[4] N. Patel et al., "Analyzing hardware based malware detectors", In DAC'17, June 2017.
[5] A. Garcia-Serrano et al., "Anomaly detection for malware identification using hardware performance counters", preprint arXiv:1508.07482, 2015.
[6] Guthaus et al., "Mibench: A free, commercially representative embedded benchmark suite", In IISWC'01, 2001.
[7] M. Hall et al., "The weka data mining software: an update", ACM SIGKDD explorations newsletter, 2009.
[8] M. Helsley, "Lxc: Linux container tools. IBM developer Works Technical Library, 2009.
[9] Intel. "Intel 64 and ia-32 architectures software developer's manual, volume 3b: System programming guide", Part, 2:18-65, 2016.
[10] G. Jacob et al., "Behavioral detection of malware: from a survey towards an established taxonomy", Journal of Computer Virology, 4(3):251{266, 2008.
[11] Kh. Khasawneh et al., "Ensemble learning for low-level hardware-supported malware detection", In RAID'15, pages 3-25. Springer, 2015.
[12] McAfee Labs. Infographic: Mcafee labs threats report. December 2017.
[13] M. Ozsoy et al., "Malware-aware processors: A framework for efficient online malware detection", In HPCA'15, 2015.
[14] H. Sayadi et al., "Machine learning-based approaches for energy-efficiency prediction and scheduling in composite cores architectures", In ICCD'17, Boston, MA, November 2017.
[15] A. Tang et al., "Unsupervised anomaly-based malware detection using hardware features", In International Workshop on Recent Advances in Intrusion Detection, pages 109-129. Springer, 2014.
[16] Wang et al., "Confirm: Detecting firmware modifications in embedded systems using hardware performance counters," In ICCAD'15, 2015.
[17] Sprunt et al., "The basics of performance-monitoring hardware", In IEEE Micro'02, November 2002.
[18] Y. Freund et al., "Schapire. A decision-theoretic generalization of on-line learning and an application to boosting", Journal of Computer and System Sciences, 55(1):119–139, August 1997.
[19] L. Breiman, "Bagging predictors", In Springer Journal of Machine Learning, Vol. 24, pp. 123–140, August 1996.
[20] H. Sayadi et al., "Scheduling multithreaded applications onto heterogeneous composite cores architecture", In IGSC'17, Orlando, FL, October 2017.
[21] C. Malone., et al., "Are hardware performance counters a cost-effective way for integrity checking of programs", In ACM STC Workshop, 2011.
[22] H. Sayadi et al., "Power conversion efficiency-aware mapping of multithreaded applications on heterogeneous architectures: A comprehensive parameter tuning" In ASP-DAC'18, South Korea, January 2018.
[23] N. C. Doyle et al., "Performance impacts and limitations of hardware memory access trace collection," In DATE'17, May 2017.
[24] H. Sayadi et al., "Comprehensive Assessment of Run-Time Hardware-Supported Malware Detection Using General and Ensemble Learning", In CF'18, Ischia, Italy, May 2018.