Comprehensive Assessment of Run-Time Hardware-Supported Malware Detection Using General and Ensemble Learning

Hossein Sayadi[1], Sai Manoj P D[1], Amir Houmansadr[2], Setareh Rafatirad[1], Houman Homayoun[1]
[1]George Mason University, Fairfax, VA, USA
[2]University of Massachusetts Amherst, Amherst, MA, USA
Email: [1]{hsayadi, spudukot, srafatir, hhomayou}@gmu.edu, [2]amir@cs.umass.edu

ABSTRACT
Recent studies have demonstrated the effectiveness of Hardware Performance Counters (HPCs) for detecting pattern of malicious applications. Hardware-supported detectors utilize Machine Learning (ML) classifiers for malware detection by analyzing a large number of HPC features, more than the very limited number of HPC registers available in modern microprocessors. Obtaining more HPCs requires running the application (malware or benign) more than once to collect the required data, which in turn makes the solution less practical for run-time detection of malware. In response to this challenge, in this work, we first identify the critical HPC features required for malware detection. Next, we explore the use of various ML techniques to classify benign and malware applications using the selected HPCs at run-time. Further, we investigate the effectiveness of ensemble learning in improving the performance of ML classifiers. For this purpose, we apply AdaBoost on all general ML classifiers. We thoroughly compare the general and ensemble ML classifiers in terms of accuracy, robustness, performance, and hardware overhead. The experimental results indicate that ensemble learning enhances the performance of malware detection for rule-based and tree-based algorithms up to 13%. However, it diminishes the performance of neural network and Bayesian network-based detectors by 6% and 4%, respectively.

KEYWORDS
Malware Detection, Machine Learning, Hardware Performance Counters, Ensemble Learning

1 INTRODUCTION
Malicious software (Malware) is a piece of program that is specifically designed with the intent of stealing or corrupting the data and/or ceasing the functionality of the systems without the consent of user. According to a 2017 McAfee threats report [13], 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. The recent proliferation of computing devices in mobile and IoT domains have also made effective detection of malware a vital challenge to address. Traditional malware detection methods such as signature-based detection and semantics-based anomaly detections are considered as software-based solutions that often incur significant computational overhead [11]. In response, the hardware-supported solutions [3,4,5,6,14] show promising results, reducing the latency of detection process by order of magnitude with small hardware cost.

Machine learning-based solutions play an important role in automated malware detection. Such malware detection method can be implemented in microprocessor hardware with significantly low overhead as compared to the software-based methods due to fast detection inside hardware [4]. These classifiers are trained using low-level features such as processor Hardware Performance Counters (HPCs) data which are captured at run-time to appropriately represent application behavior. HPCs are a set of special-purpose registers built in modern microprocessors to capture the count of hardware-related events which have been extensively used to predict the power, performance, and energy efficiency of computing systems [9,15,16]. Recent studies have demonstrated that malware behavior can be differentiated from benign applications by classifying anomalies in low-level feature spaces such as micro-architectural events collected by HPC registers available in today's processors [2,3,4,5,6,11,12,14]. Malware detection using HPCs has emerged as a promising alternative for traditional malware detection methods against increasing security threats.

Previous work performed a limited study on hardware malware detection assuming the availability of a large number (e.g. 16 or 32) and diverse HPCs, whereas modern processors, especially in embedded mobile and IoT domains, have a limited number of HPCs ranging between 2 to 8. Therefore, on real systems, collecting a variety of HPC data to achieve high accuracy using the general ML model presented in prior work [3,4] requires running the program several times, since the hardware can only count a small subset of events simultaneously which makes it impractical for run-time detection of malware. In addition, prior studies mostly focus on specific classifiers ignoring the analysis of various type of ML solutions [3,11,12,13,14].

In this work, through a systematic approach, we first analyze a large number of HPCs and identify the most critical low-level features related to malware detection. Next, we thoroughly assess and characterize various type of machine learning methods to classify benign and malware applications at run-time. In addition, we investigate the effectiveness of ensemble learning in improving the performance of ML classifiers. We examine different general and ensemble learning techniques in terms of accuracy, robustness, performance, and hardware implementation costs. In contrast to prior studies, the proposed machine learning-based framework effectively performs malware detection during run-time using a limited number of HPCs. Based on the achieved performance, area, and latency metrics, we provide valuable insights that aid in choosing accurate and hardware-suitable ML classifiers for malware detection. This comprehensive analysis

CF '18, May 8–10, 2018, Ischia, Italy

ACM ISBN 978-1-4503-5761-6/18/05…$15.00
https://doi.org/10.1145/3203217.3203264

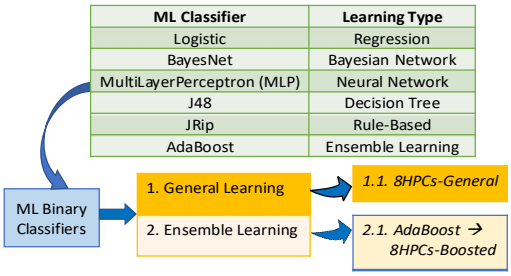

Figure 1. Machine learning classifiers used for malware detection

helps the designers to understand and navigate the trade-offs between several design parameters offered by each ML classifier.

2 PROPOSED HARDWARE-SUPPORTED MALWARE DETECTION FRAMEWORK

In this section, we present the details of our proposed approach for distinguishing the malware from benign applications.

2.1 Machine Learning Techniques

Methods of machine learning, which build predictive models that generalize training data, have proven to be useful for detecting malware. In this work, we use machine learning-based solutions for hardware-supported malware detection which rely on the run-time traces collected from HPC registers. We deploy various ML classifiers for malware detection that are shown in Figure 1. The rationale for selecting these six machine learning models are: First, they are from different branches of ML methods; regression, Bayesian network, 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.

Ensemble learning is a machine learning method which is used to improve the accuracy and performance of general learning classifiers by generating a set of base learners and then combining outputs of these base learners for a final decision. Here we deploy a well-known ensemble learning method, AdaBoost (Adaptive Boosting) to construct the final classifier and analyze its impact on the accuracy and performance improvement of malware detection. 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. All base classifiers are trained and combined to generate the final classifier. Each training sample in the dataset is weighted and the weights are updated based on the overall performance 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.

2.2 Experimental Setup and Data Collection

All applications are executed on an Intel Haswell Core i5-4590 machine running Ubuntu 14.04 with Linux 4.4 Kernel and various HPCs are collected using *Perf* tool. We have executed more than 100 benign and malware applications for HPC data collection. Benign applications include MiBench benchmark suite [7], Linux system programs, browsers, text editors, and word processor. For malware applications, Linux malware is collected from virustotal.com [1]. Malware applications consist of various classes including viruses, worms, rootkits, and trojans. After collecting micro-architectural events using *Perf*, we use WEKA tool [8] for evaluating accuracy and performance of different ML classifiers.

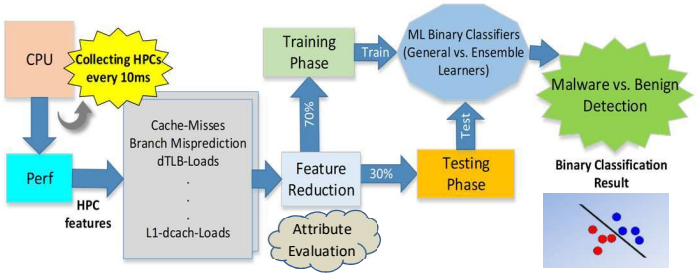

Figure 2. Overview of proposed malware detection framework

HPC information is collected by running all applications in Linux Containers (LXC) which is an isolated environment. LXC is an operating system level virtualization method that shares the same kernel with the host operating system. We extracted 44 CPU events available under *Perf* tool. Since Intel Haswell has only 8 counter registers available [10], we can only measure 8 events at a time. As a result, multiple runs are required to fully capture all events. Running malware inside the container can contaminate the environment which may affect subsequent data collection. To make sure that there is no contamination in collected data due to the previous run, we destroy the container after each run.

2.3 Overview of the Proposed Approach

Figure 2 depicts the overview of the proposed run-time hardware-supported malware detection framework and the training and testing process to build ML classifiers for predicting the malicious behavior of applications. It 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 hardware performance counters using effective correlation analysis and feature reduction methods, and developing a machine learning model from the training data. Note that the input variables in our classifiers are HPC events of the running applications at every 10ms interval, and the output is the class of an application (malware or benign). In order to validate each of our ML classifiers, we follow standard 70%-30% dataset split for training and testing. To follow a complete non-biased splitting, we separate 70% benign- 70% malware applications for training (known applications) and 30% benign- 30% malware applications for testing (unknown applications).

2.3.1 Feature Reduction. Representing programs with low-level micro-architectural features may produce very high dimensional dataset. Running ML algorithms with large HPCs data would be complex and slow. Besides, incorporating irrelevant features would lower the accuracy of the classifier. Therefore, irrelevant features are identified and discarded using a feature reduction algorithm and a subset that includes the most important features is selected and supplied to each ML classifier. The learning algorithm attempts to find a correlation between the feature values and the application behavior to detect the malware or benign type. We use *Correlation Attribute Evaluation* as a feature reduction technique on our training set under WEKA to monitor the most vital micro-architectural features and capture application characteristics. We determine the eight most related performance counters which is the maximum number of HPCs

Table 1. Critical HPCs in order of importance

1- branch instructions	2- branch_loads
3- iTLB_load_misses	4- dTLB_load_misses
5- dTLB_store_misses	6- LLC_prefetch_misses
7- L1_dcache_stores	8- cache_misses

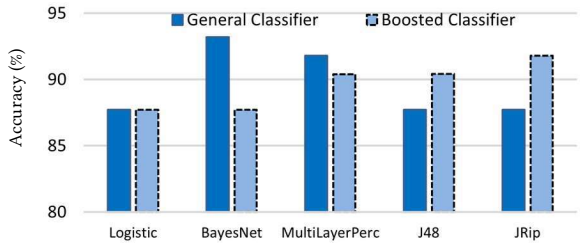

Figure 3. Accuracy results for various ML classifiers with 8 HPCs

collected simultaneously in today's microprocessors. These HPCs are listed in Table 1 and numbered in order of their importance in malware detection. They are included in our ML detectors as input parameters. These features include HPCs representing pipeline front-end, pipeline back-end, cache subsystem, and main memory behaviors and are influential in the performance of standard applications.

3 EVALUATION RESULTS

In this section, we present the evaluation results of the ML-based malware detectors.

3.1 Accuracy Analysis of Detectors

To evaluate the malware detection accuracy of ML classifiers, we consider the percentage of correctly classified samples. Figure 3 shows the malware detection accuracy of various ML classifiers (general vs. AdaBoost) using eight HPCs. AdaBoost improves the performance of malware detection for weak classifiers such as JRip and J48. However, it shows negative impact on strong classifiers such as MLP and BayesNet. Specifically, as seen in Figure 3, for JRip (rule-based classifier) and J48 (tree-based classifier) by applying AdaBoost ensemble learning, the malware detection accuracy is improved by 4% and 5%, respectively, as compared to their base classifiers. On average an accuracy of 87.7% is achieved with general base learners which is improved to 91.2% with AdaBoost for lightweight classifiers such as J48 and JRip when only 8 HPCs are employed.

Contrarily, up to 6% accuracy degradation is observed when AdaBoost is applied on complex classifiers such as MLP (neural network) and BayesNet (Bayesian network) compared to their base models. In addition, as can be seen, BayesNet classifier without boosting achieves the highest detection accuracy of 93.5% which is higher than AdaBoost implementations of weak classifiers (JRip and J48). The detection accuracy for general MLP-based detector is 92% which is close to Boosted-JRip and 2% higher than the Boosted-J48 algorithm.

These observations confirm the effectiveness of using ensemble learning to boost the accuracy of rule-based and decision tree classifiers in comparison with using them for heavyweight classifiers such as MLP and BayesNet. For instance, as shown, JRip itself achieves close to 88% accuracy with 8 HPCs. However, we observe that constructing AdaBoost model with the base learner of JRip results in achieving almost 92% accuracy. Moreover, boosting leads in reducing the accuracy in MLP and BayesNet general classifiers indicating that AdaBoost technique is best suited for weak rule-based and tree-based classifiers and using it for a neural network and Bayesian models diminishes the accuracy of malware detection.

3.2 Robustness of Detectors

To evaluate the robustness of ML classifiers in detecting malware, Receiver Operating Characteristics (ROC) graphs are used. Robustness is referred to how well the classifier distinguishes between binary malware and benign classes. The ROC curve is

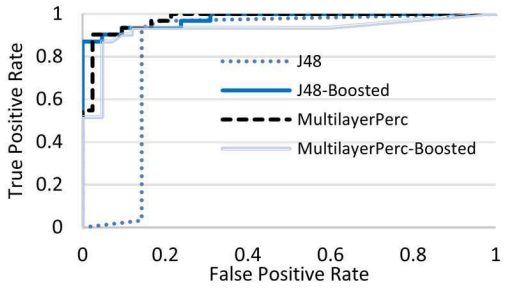

Figure 4. ROC graphs of four general and ensemble classifiers

produced by plotting the fraction of true positives rate versus the fraction of false positives for a binary classifier. We use the Area Under the Curve (AUC) measure for ROC in the evaluation process to examine the robustness of each ML classifier. The AUC value of the best possible classifier is equal to meaning that a discrimination threshold can be found under which the classifier achieves 0% false positives and 100% true positives.

Figure 4 depicts the ROC for four different ML-based malware detectors. Due to the space limitation, we only show the ROC graphs for selected ML classifiers. In this figure, the ROC graphs for general J48 and MLP, as well as the corresponding AdaBoost models are shown, considering 8 HPCs. Similar to the accuracy of malware detection, the robustness of weak classifiers (such as J48) are improved by 14% with boosting making the J48-Boosted more effective in terms of classification robustness, whereas the robustness of strong classifiers such as MLP is reduced by 6%. In addition, as observed in Figure 4, the Boosted-J48 delivers the AUC of equal to the general MLP (both 0.98). This also indicates that weak classifiers combined with ensemble learning can match the robustness of the general strong classifiers eliminating the need to use complex classifiers for better accuracy and robustness.

3.3 Performance Analysis of Detectors

To simultaneously evaluate both accuracy and robustness of different ML classifiers for malware detection, we employ the product of accuracy and area under the ROC graph (ACC*AUC) as a performance metric. Figure 5 depicts the performance of various general and boosted ML classifiers. First of all, regardless of boosting effect, the results indicate that MLP outperforms the performance of boosted weak classifiers delivering the highest performance of 90% for malware detection. Next in the general classifiers, the BayesNet achieves a performance of close to 87%. Similar to previous observations, out of the five different ML classifiers, lightweight classifiers (J48 and JRip) are benefiting from boosting approach delivering higher performance (13% and 9%, respectively, as compared to their general model) when used as a base learner in AdaBoost ensemble technique. J48 and JRip are decision trees and rule-based algorithms, respectively with reasonably fast training and classification process making them suitable fits for boosting techniques. This shows a potential for

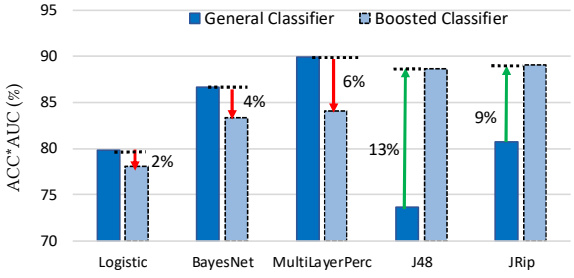

Figure 5. Performance of ML classifiers with 8 HPCs

Table 2. Hardware implementation results

	General		AdaBoost	
Classifier	**Latency**	**Area**	**Latency**	**Area**
Logistic	68	13041	102	23762
BayesNet	14	6794	56	10345
MLP	302	36252	591	47861
J48	9	1801	67	2589
JRip	4	1504	56	3192

applying ensemble learning techniques to boost the accuracy and performance of detectors. It is a crucial benefit when using such classifiers for base leaners of boosting learning where we need to run a model multiple times before outputting the final decision.

On the other hand, it is evident from the figure that applying AdaBoost on heavyweight classifiers (MLP, BayesNet, and Logistic) have negative impact on their performances. As seen, using AdaBoost technique reduces the performance of MLP the most by 6%. Next impacted classifier is BayesNet by 4% performance reduction and Logistic regression-based classifier is least influenced by AdaBoost among the strong classifiers. On an average, the weak classifiers achieve an improvement of 11% in performance. In contrast, averagely 4% reduction is observed in performance of heavyweight classifiers with AdaBoost. The results clearly demonstrate the direct dependency of AdaBoost ensemble technique's performance on the type of its base ML classifier.

3.4 Hardware Overhead Analysis

When it comes to choosing ML classifiers for hardware implementation, the accuracy of an algorithm is not the sole contributor for decision-making. Design area and response time (latency) of ML classifiers are also key factors in selecting the cost-efficient solution. For hardware implementation, we deploy Vivado HLS compiler to synthesize ML classifiers for Xilinx Virtex-7 FPGA. Table 2 provides the hardware implementation costs for general classifiers and ensemble method (AdaBoost) applied on each general classifier using 8 HPCs. Latency unit is in terms of number of clock cycles (@10 ns) required to classify input. Area unit is the total number of utilized LUTs, FFs, and DSP units inside Virtex-7 FPGA. As depicted, the MLP and Logistic Regression result in significant area and latency overhead, compared to other models, while JRip and J48 produce lowest area overhead and latency among all implemented classifiers. Besides, as expected AdaBoost learning introduces significant area overhead across different ML classifiers. However, the area required for the Boosted lightweight classifiers (JRip and J48) is still smaller than the area required for the general heavyweight classifiers, and a same trend is observed in terms of latency.

To accordingly account for accuracy, robustness and area overhead, in Figure 6 we compare performance over a unit of hardware area, (ACC*AUC)/Area, for various ML classifiers. We propose using performance over area to identify classifiers that require small area and yet can detect the maliciousness of program with high accuracy and robustness. A classifier with a higher ratio is considered better than the ones with lower ratio. Among all classifiers, the rule-based and tree-based classifiers are found to be more efficient compared to the highly accurate but complex BayesNet, MLP, and Logistic classifiers. Although by applying AdaBoost the performance/area is slightly reduced in JRip and J48 algorithms, it is still higher than other costly classifiers. The MLP has the least performance per unit area due to its large area overhead, whereas the JRip has the highest performance/area among the general ML classifiers. Despite the area overhead caused by boosting for weak classifiers, the performance per unit area is higher than other experimented

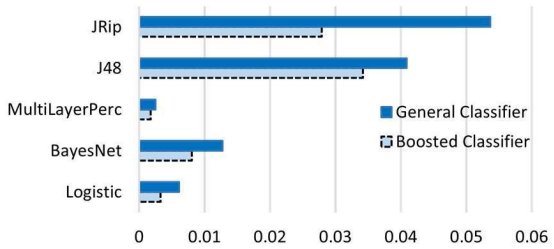

Figure 6. Performance/Area comparison of various ML classifiers

heavyweight classifiers. Clearly, the results show some trade-offs between accuracy, performance, and area overhead. Therefore, it is important to compare ML classifiers for malware detection by taking all these parameters into consideration.

4 CONCLUSION

In this work, we proposed a run-time hardware-supported malware detection framework which effectively detects the malicious software with limited number of HPCs. We implemented various learning models including general ML classifiers and AdaBoost ensemble learning and thoroughly evaluated them in terms of accuracy, robustness, performance, and hardware overhead. We showed that using eight HPCs can deliver sufficient accuracy of nearly 94% for effective run-time malware detection. The results indicate that without hardware overheads consideration, complex ML classifiers such as MLP and BayesNet are the winners given their higher performance. However, after accounting for the implementation costs, they perform worst in terms of performance/area and latency compared to significantly simpler but slightly less accurate classifiers such as JRip and J48. Also, the rule-based and tree-based techniques by showing up to 13% performance improvement benefit more from application of ensemble learning. These lightweight classifiers combined with ensemble learning can match the robustness of the general strong classifiers eliminating the need to use costly classifiers for better performance.

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 ICCKE'14, 2014.

[3] Demme et al., "On the feasibility of online malware detection with performance counters", In ISCA'13, 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] H. Sayadi et al., "Ensemble learning for effective run-time hardware-based malware detection: A comprehensive analysis and classification", In DAC'18, San Francisco, CA, June 2018.

[7] Guthaus et al., "Mibench: A free, commercially representative embedded benchmark suite", In IISWC'01, 2001.

[8] M. Hall et al., "The weka data mining software: an update", ACM SIGKDD explorations newsletter, 2009.

[9] H. Sayadi et al., "Machine learning-based approaches for energy-efficiency prediction and scheduling in composite cores architectures," In ICCD'17, November 2017.

[10] Intel. "Intel 64 and ia-32 architectures software developer's manual, volume 3b: System programming guide", Part 2:18-65, 2016.

[11] Jacob et al., "Behavioral detection of malware: From a survey towards an established taxonomy",Journal in Computer Virology, 2008.

[12] Kh. Khasawneh et al., "Ensemble learning for low-level hardware-supported malware detection", In RAID'15, pp. 3-25. Springer, 2015.

[13] McAfee Labs. Infographic: Mcafee labs threats report. December 2017.

[14] M. Ozsoy et al., "Malware-aware processors: A framework for efficient online malware detection", In HPCA'15, 2015.

[15] H. Sayadi et al., "Power Conversion Efficiency-Aware Mapping of Multithreaded Applications on Heterogeneous Architectures: A Comprehensive Parameter Tuning" In ASP-DAC'18, January 2018.

[16] H. Sayadi et al., "Scheduling multithreaded applications onto heterogeneous composite cores architecture," In IGSC'17, October 2017.