Can We Reliably Detect Malware Using Hardware Performance Counters?

Boyou Zhou, Anmol Gupta, Rasoul Jahanshahi, Manuel Egele, Ajay Joshi
Eletrical and Computer Engineering Department, Boston University
(bobzhou, anmol.gupta1005, rasoulj, megele, joshi)@bu.edu

ABSTRACT

The ever-increasing prevalence of malware has led to the explorations of various detection mechanisms. Several recent works propose to use Hardware Performance Counters (HPCs) traces with machine learning classification models for malware detection. HPCs are hardware units that record low-level micro-architectural events, such as cache hits/misses, branch (mis)prediction, and load/store operations. However, information about these events does not capture the semantic behaviors of the application. In our paper[16], we claim and experimentally justify that using the micro-architectural level information obtained from HPCs cannot distinguish between benignware and malware. We harvest the HPC traces from 1,924 programs, 962 benignware, and 962 malware, on our experimental setups. We achieve an F1-score (a metric of detection rates) of 83.39%, 84.84%, 83.59%, 75.01%, 78.75%, and 14.32% for Decision Tree (DT), Random Forest (RF), K Nearest Neighbors (KNN), Adaboost, Neural Net (NN), and Naive Bayes, respectively. We cross-validate our models 1,000 times and the F1-score of models in DT, RF, KNN, Adaboost, NN, and Naive Bayes is 80.22%, 81.29%, 80.22%, 70.32%, 35.66%, and 9.903%, respectively. To show how fragile the HPC malware detection system is, we show that one benignware (Notepad++) infused with malware (ransomware) cannot be detected by HPC-based malware detection.

1 BACKGROUND AND MOTIVATION

Detecting malware is an ongoing challenge for security researchers. Over the past several years, security researchers have developed many malware detection techniques, such as signature-based malware analysis and behavior analysis. These software techniques extract semantic information from the program to detect malware. However, software techniques inherently introduce overheads in the process.

To overcome this degradation in program performance, many researchers have proposed to use Hardware Performance Counters (HPCs) to capture micro-architecture events and apply machine learning (ML) algorithms to classify the programs as malware or benignware [3, 5–7, 11, 14, 15]. Prior HPC malware detections follow a general pattern: they performed fixed-frequency HPC measurements on a pool of existing malware and benignware, aggregated the measured HPC traces in a time series, extracted features. Then previous works apply the extracted features in various ML algorithms to generate classification models. At runtime, the detection systems measured HPC traces and applied the pre-trained models to classify the monitored program as malware or benignware.

The approach of malware detection using HPCs relies on the assumption that the maliciousness of the program affects the measured HPC traces. However, this assumption is counter-intuitive. For example, consider a password manager program and a ransomware program. Both programs use cryptographic APIs and upload the keys to the remote server. The program logic of the password manager and the ransomware are the same. The only difference is the location of the remote server. One cannot distinguish between the two locations based on the traces of HPCs because there is no micro-architecture event that distinguishes the two servers based on their IP addresses.

Moreover, it is expected from previous works to provide rigorous analysis to justify *how* and *why* to choose these micro-architectural events for malware detection. However, all previous works elide this discussion and commit the logical fallacy of "cum hoc ergo propter hoc" – concluding causation from correlation. The results in the previous work are based on limited program samples and experiments that put the detection system at an advantage.

In our paper[16], we evaluate the proposed idea of detecting malware with HPCs using a realistic setup and performing a comprehensive and rigorous analysis. Our analyses show that the experiments conducted in previous works are based on unrealistic assumptions and insufficient analyses. We classify the drawbacks of the prior works into the following categories:

1) Using Dynamic Binary Instrumentation tools (DBI) to gather HPC traces - Hardware vendors usually provide software APIs to sample HPC values. DBI tools such as Intel's Pin [9], QEMU [2], Valgrind [10], and DynamoRIO [1], can monitor micro-architectural events in a processor. Khasawneh et al. use these tools in their experiments to extract the HPC traces and detect malware [6, 7, 11]. However, DBI tools introduce as much as 10× performance overhead during the program execution, which is prohibitive in online malware detection.

2) Performing the experiments on Virtual Machines (VMs) - Besides DBI tools, some works use VMs for sampling HPC values in the experiments [3, 6, 7, 11, 13]. VMs leverage virtualized HPC technology to extract the HPC values from the host machine. Reliably virtualizing HPCs is a challenge [12] and it is nearly impossible to measure the HPCs accurately from VMs. There exists malware that detects VM environment to evade detection [8]. Hence, experiments on VMs cannot conclusively say that HPCs can be used to detect malware.

3) Divisions of the training-testing dataset based on HPC traces - Our literature survey revealed that prior works divide the training and testing datasets based on the *measured HPC traces* (Training-testing Approach 1, **TTA1**) [3, 5, 13, 15], instead of division based on *program samples* (Training-testing Approach 2, **TTA2**). In **TTA1**, traces measured from the same program sample exist in both training and testing dataset. **TTA1** represents a scenario where we can train the ML model using all the binaries that a user will ever encounter. This is highly unrealistic in real life

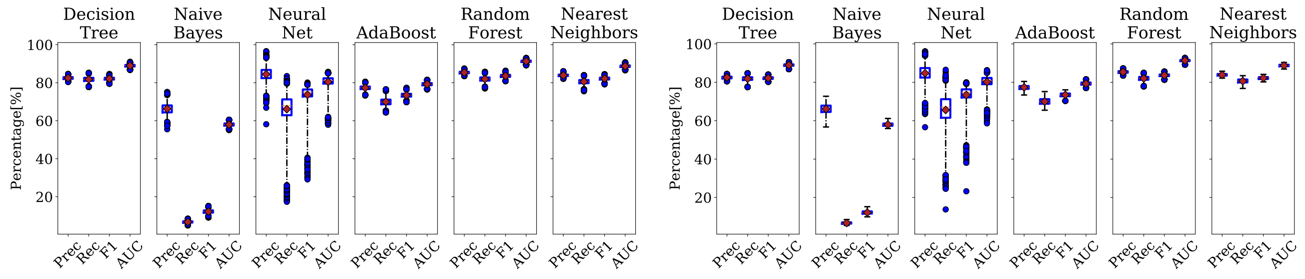

Figure 1: Box plots of distributions of 10-fold cross-validation experiments using (a) TTA1 and (b) TTA2.

because polymorphic malware can generate an infinite number of forms with the same original binary.

4) No cross-validation or insufficient cross-validation - When developing an ML model, it is recommended that one trains and tests the ML model multiple times by partitioning the available data set differently every time. This cross-validation approach prevents the ML model from overfitting[1]. It is recommended by the ML community to use 10-fold cross-validation to evaluate a given classifier [4]. Some of the previous works have no cross-validation [3, 14, 15], while others have insufficient cross-validation [6, 7, 11, 13] of the trained models. In our experiments, the overall detection rates using **TTA2** have much higher variations compared ones using **TTA1** in Figure 1.

5) Program counts used for training and testing are unrealistic - Previous works have performed their evaluation using a dozen program samples [3, 5, 13–15]. With two decimal precision and ten cross-validations in the reported results, the experiment requires at least 1,000 program samples. In order to be accurate with the reported precision, the results should have either less decimal precision or more program samples.

2 KEY CONTRIBUTIONS OF THE PAPER

To rigorously evaluate the proposed idea of detecting malware using HPCs, we performed our experiments with 1,924 program samples on a cluster of 16 bare-metal AMD machines. Identical to prior work in this area, after gathering the HPC traces, we aggregated the measured values in the traces into 32 bins, and then performed Principal Component Analysis (PCA) to reduce the data dimensions for training, and testing. Unlike prior work, we applied 10-fold cross-validation. In our experiments, we observed a False Discovery Rate[2] of more than 20%. This would mean that in a default Windows 7 installation with 1,323 executable files, 264 files would get incorrectly classified as malware.

To show how fragile the HPC malware detection is, we built a **simple** malware that is composed of a benign program (Notepad++) with malicious functionality (ransomware). While a classifier accurately distinguished the benign and malicious programs in isolation, the combination of two source-code bases resulted in a malicious version of Notepad++ that the classifier incorrectly labeled as benignware (i.e., a false negative).

[1]ML model fits closely to one dataset but fails in the others.
[2]$F_+/(F_+ + T_+)$, where F_+ is the number of benignware classified as malware and T_+ is the number of malware classified as malware

In summary, our paper[16] makes the following contributions:

- We identify the prevalent unrealistic assumptions and insufficient analyses used in prior works that leverage HPCs for malware detection.
- We perform a rigorous experimental evaluation with a program count that exceeds previous works [3, 5, 13–15] by a factor of $2\times \sim 3\times$, and the number of experiments in cross-validations that is three orders of magnitude more than previous works.
- We train and test dataset using two training-testing dataset division approaches – division based on measured HPC traces (**TTA1**) and division based on program samples (**TTA2**). We compare the quality of the ML models for these two choices and demonstrate that ML models based on realistic user setting, i.e. **TTA2**, can lead to a FDR of 20%.
- Finally, in the spirit of open science, we make all our code, data, and result publicly available under an open-source license: https://github.com/bu-icsg/Hardware_Performance_Counters_Can_Detect_Malware_Myth_or_Fact

REFERENCES

[1] 2017. DynamoRIO. http://www.dynamorio.org/. (2017). (Accessed on 12/02/2017).
[2] Fabrice Bellard. 2005. QEMU, a fast and portable dynamic translator.. In *ATC*.
[3] John Demme et al. 2013. On the feasibility of online malware detection with performance counters. In *ISCA*. 559.
[4] Ian Goodfellow et al. 2016. *Deep learning*. MIT press.
[5] Mikhail Kazdagli et al. 2016. Quantifying and improving the efficiency of hardware-based mobile malware detectors. In *MICRO*. 1–13.
[6] Khaled N Khasawneh et al. 2015. Ensemble learning for low-level hardware-supported malware detection. In *RAID*.
[7] Khaled N Khasawneh et al. 2017. RHMD: evasion-resilient hardware malware detectors. In *MICRO*. 315–327.
[8] Dhilung Kirat et al. 2014. BareCloud: Bare-metal Analysis-based Evasive Malware Detection. In *SP*.
[9] Chi-Keung Luk et al. 2005. Pin: building customized program analysis tools with dynamic instrumentation. In *SIGPLAN*. ACM.
[10] Nicholas Nethercote et al. 2007. Valgrind: a framework for heavyweight dynamic binary instrumentation. In *SIGPLAN*.
[11] Meltem Ozsoy et al. 2015. Malware-aware processors: A framework for efficient online malware detection. In *HPCA*. 651–661.
[12] Benjamin Serebrin et al. 2011. Virtualizing performance counters. In *ECPP*.
[13] Baljit Singh et al. 2017. On the detection of kernel-level rootkits using hardware performance counters. In *AsiaCCS*. ACM, 483–493.
[14] Adrian Tang et al. 2014. Unsupervised anomaly-based malware detection using hardware features. In *RAID*. 109–129.
[15] Xueyang Wang et al. 2016. Hardware Performance Counter-Based Malware Identification and Detection with Adaptive Compressive Sensing. *TACO* (2016).
[16] Boyou Zhou et al. 2018. Hardware Performance Counters Can Detect Malware: Myth or Fact?. In *AsiaCCS*.