HMD-Hardener: Adversarially Robust and Efficient Hardware-Assisted Runtime Malware Detection

Abhijitt Dhavlle*, Sanket Shukla*, Setareh Rafatirad†, Houman Homayoun†, Sai Manoj Pudukotai Dinakarrao*
*George Mason University, Fairfax, VA, USA.
†University of California Davis, Davis, CA, USA.
Email: {adhavlle, sshukla4, spudukot}@gmu.edu, {srafatirad, hhomayoun}@ucdavis.edu

2021 Design, Automation & Test in Europe Conference & Exhibition (DATE) | 978-3-9819263-5-4/21/$31.00 ©2021110.23919/DATE51398.2021.9474036

***Abstract*—To overcome the performance overheads incurred by the traditional software-based malware detection techniques, machine learning (ML) based Hardware-assisted Malware Detection (HMD) has emerged as a panacea to detect malicious applications and provide security. HMD primarily relies on the generated low-level microarchitectural events captured through Hardware Performance Counters (HPCs). This work proposes an adversarial attack on the HMD systems to tamper the security by introducing perturbations in performance counter traces with an adversarial sample generator application. To craft the attack, we first deploy an adversarial sample predictor to predict the adversarial HPC pattern for a given application to be misclassified by the deployed ML classifier in the HMD. Further, as the attacker has no direct access to manipulate the HPCs generated during runtime, based on the adversarial sample predictor's output, devise an adversarial sample generator wrapped around the victim application to produce HPC patterns similar to the adversarial predictor's estimated trace. With the proposed attack, malware detection accuracy is reduced to 18.1% from 82%. To render the HMD robust to such attacks, we further propose adversarially training the HMD to demonstrate that hardening can render HMD resilient against attacks; the detection accuracy post hardening raises to 81.2%.**

I. Introduction

The ever-increasing complexity of modern computing systems has resulted in the growth of security vulnerabilities, making such systems an appealing target for sophisticated attacks. Computing systems today are employed to deliver high performance and efficiency while protecting users' data. Attackers take advantage of malware's capabilities to harm the system, steal users' data, disrupt the system, or a combination of it. Malware, also known as malicious software, is a program or application to infect the computing systems without the user agreement for serving harmful purposes such as stealing sensitive information, unauthorized data access, destroying files, running intrusive programs on devices to perform Denial-of-Service attack, and disrupting essential services.

A plethora of works focus on detecting malware, but the downside of using software-based approaches is the overhead, owing to computational complexity. Also, software-based detection utilizes signature-based detection that matches the behavior signature of the application to its database. This approach fails to recognize zero-day attacks, and signatures that do not match its database, given an outdated database. We focus on hardware-based detection approach.

To overcome shortcomings such as latency and computational complexity of traditional malware detection techniques, including signature and semantics-based software-driven techniques [1], [2], Hardware-Assisted Malware detection (HMD) approaches are proposed [3]. HMD refers to utilizing the low-level microarchitectural hardware events for detecting and classifying the malware from benign applications. Browsers, utility applications, text editors, C-based programs were some of the benign applications that we profiled as a part of the dataset. In contrast, applications embedded with Trojan, worm, virus, and other malwares were profiled as part of the malign traces. The HMD delivers reduced malware detection latency by orders of magnitude with smaller hardware cost [3].

This work proposes an adversarial attack on HMDs in which the adversarial samples are generated through a benign code that is wrapped around a benign or malware application to produce a desired output class from the embedded ML-based malware detector. One of the main challenges to address is that the attacker or user has no direct access to modify the HPC and manipulation of HPCs is highly complex to perform despite employing techniques like code obfuscation for executing malware [3]. First, we assume the victim's defense system is unknown and perform reverse engineering to mimic the embedded HMD's behavior and build a machine learning (ML) classifier. To determine the required number of HPCs to be generated through the application to be misclassified, we employ an '*adversarial sample predictor*' which predicts the number of HPC traces to be generated to misclassify an application by the HMD. by the attacker. To perturb the HPCs, we craft an '*adversarial HPC generator*' application (code) that generates the required number of HPCs. This adversarial HPC generator application is wrapped around the application that needs to be misclassified by perturbation.

This work discusses a novel way of crafting adversarial HPC traces through a benign application and proposes a methodology on how to craft such an application to obtain adversarial behavior. The main focus of this work is to generate false alarms (malware classified as benign and benign classified as malware) to weaken the trust on the embedded defenses, which increases the scope for attacks. The proposed work benefits from the following: a) no need to tamper or modify the source code of the application around which the proposed adversarial sample generator code is wrapped; b) the crafted application has no malicious features embedded, thus not detectable by malware detectors; c) scalable and flexible, i.e., the crafted application can generate events required to create a powerful adversary. With these adversarial attacks, the HMD delivers unacceptable performance. To make the HMD robust and resilient to such adversarial attacks, we propose to perform adversarial

learning by training the HMD on adversarial samples. This method has proven successful for different types of adversarial attacks and can boost the HMD performance to classify malign applications from their benign counterparts reliably. We then present hardware implementation of the ML classifiers used in the HMD for analysis purposes.

II. State-Of-The-Art

Recent works [3]–[10] have shown that by deploying Machine Learning (ML) techniques fed with the low-level microarchitectural events (features) captured by Hardware Performance Counters (HPCs) can aid in differentiating benign and malware applications. HPCs were introduced to aid debugging but were exploited later by attackers and their counterparts. Performance counters are a set of special-purpose registers built into modern microprocessors to capture the trace of hardware-related events such as LLC misses, branch instructions, branch misses, and executed instructions while executing an application.

The work in [3] was one of the preliminary works proposed to utilize the HPC data for malware detection. It demonstrated the effectiveness of offline ML algorithms in malware classification. The researchers in [11] and [4] discuss the feasibility of employing unsupervised learning method on low-level features to detect Return-Oriented programming (ROP) and buffer overflow attacks by finding an anomaly in the information received from the hardware performance counters. The work in [12] uses logistic regression to classify malware into multiple classes and train a specialized classifier for detecting malware class. The ML-based malware detectors (HMD) can be implemented in a microprocessor hardware with significantly low overhead compared to the software-based methods, as detection inside the hardware is very fast (few clock cycles) [2]. In summary, it is seen that a large body of works have been dedicated to employing low-level microarchitectural events fed to ML classifiers to make the systems secure.

On the other hand, despite the ML classifiers being deployed in numerous applications and shown robustness against random noises, the exposed vulnerabilities have shown that the outcome of ML classifiers can be modified or controlled by adding specially crafted perturbations to the input data [13]–[18], often referred to as *Adversarial samples*. A plethora of works on adversarial attacks exist, focusing specifically on computer vision applications [13]–[16], where the number of features are often substantial. Recently, a few works on crafting adversarial malware are as well proposed in [19]. However, works such as [19] consider the application features in a binary format (feature exists or not) for showcasing the attack and defense. Though the application features (in binary format) are manipulated, traditional techniques such as semantic and signature analysis based methods can detect these adversaries [20]. Similarly, authors in [21] present the efficiency of detecting malware through HPCs. Though the presented experimental results in [21] are in favor of efficient malware detection through HPCs, they claim that if HPC traces of malware and benign applications are similar, it is hard to detect malware. However, no details on crafting such malware are provided, limiting its practical application.

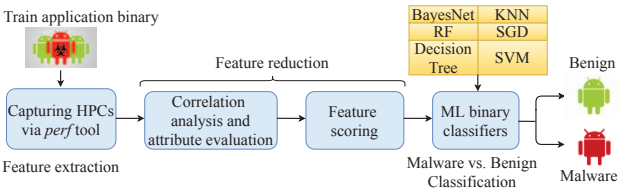

Fig. 1: Overview of the proposed hardware-based malware detection process

III. Hardware Assisted Malware Detection

This section briefly discusses the overview of hardware-assisted malware detectors, referring to Figure 1. The preliminary step in the training process requires profiling applications and generate a dataset that can be later used for training machine learning models - the heart of the HMD. The dataset comprises captured features that describe the hardware's microarchitectural state at different time instances for applications executing on the system. The process of profiling of applications is described in the figure as the "Feature Extraction" block. These HPCs are used to build the dataset comprising all the applications with the corresponding features (performance counters). We train and deploy multiple ML-models in the HMD to observe the model that delivers the best performance in detecting malware.

TABLE I: Microarchitectural events important for runtime malware detection

Rank	Event name	Rank	Event name
1	Branch Instructions	5	dTLB_store_misses
2	Branch Loads	6	LLC_prefetch_misses
3	iTLB_load_misses	7	L1_dache_stores
4	dTLB_load_misses	8	cache_misses

A. Feature Selection

Representing programs at a low microarchitectural level produces a very high dimension dataset. Running ML algorithms with a large dataset are complex and slow. Therefore, instead of considering all captured features, irrelevant data is identified and removed using a feature reduction algorithm. A subset of captured traces is selected that includes the most important features for classification. We collected 44 performance counters, as they were all the hardware events our experimental setup allowed. We first use Correlation Attribute Evaluation on our training set 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. The eight most related hardware performance counters are determined and numbered in order of importance for malware detection by applying the feature reduction method. These HPCs are listed in Table I. Most modern processors allow recording 4 or 8 events simultaneously. Hence, to suit the detection process, given the hardware limitation on the number of events that can be collected, we constant the feature size to four. The HMD delivers high performance in detecting malware with four features, and collecting four features in runtime generates less overhead compared to eight.

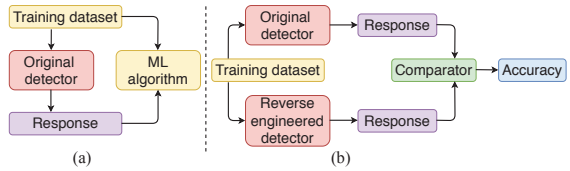

Fig. 2: (a) Process of reverse engineering an HMD; (b) Testing Performance of Reverse-Engineered Detector

B. Training & Testing the Malware Detectors

The training/testing of the HMD involves feeding the previously selected HPCs (performance counters) in runtime. It is important to note that the input variables in our classifiers are the HPCs extracted every 10 ms interval from the running applications, and the output variable is the class (malware or benign) of an application. The 10 ms sampling interval is chosen based on the frequency versus overhead trade-off. For each ML classifier, we construct general ML models 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. The accuracy of each ML classifier is presented in Section VII.

IV. Adversarial Sample Prediction

In this section, we discuss the adversarial attack on HMD, our proposed adversarial sample predictor, and the adversarial sample generator to degrade HMD performance.

A. Adversarial Attacks on ML-Classifiers

In this work, the terms 'adversarial malware' and 'adversarial attack' are used interchangeably. Similarly, 'adversarial defense' and 'hardening' have been used interchangeably. Works in [14], [22] describe the process of different adversarial attacks on ML classifiers. The fundamental idea is to perturb the inputs such that the performance of the classifier is degraded. We propose to exploit the concept of input sample perturbation to attack the HMDs, as will be discussed further.

B. Reverse Engineering an HMD

Under the assumption where the victim malware detector is unknown, we perform reverse engineering to mimic the functionality of the victim[1] HMD detector. Thus, as a first step to craft adversarial malware, we perform reverse engineering of the victim's HMD similar to that proposed in [20]. The performed reverse engineering is described in Figure 2.

We first create a training dataset that comprises benign and malware applications. Nearly 10,000 benign and 10,000 malware applications are used in the reverse engineering process. The victim's HMD (Original HMD) is fed with all the applications, and the responses are recorded. These responses are utilized for training different ML classifiers to mimic the functionality of the victim's HMD, as shown in Figure 2(a). Further, it is tested by comparing the outputs from the victim's HMD response and the reverse-engineered ML classifier's response, as shown in Figure 2(b). Reverse engineering is non-trivial as the adversaries generated on a closely functional model will be highly effective compared to a weakly developed adversary. To ensure reverse engineering is performed efficiently, we train multiple ML classifiers and choose the classifier that yields high performance, i.e., mimics the victim's HMD with high accuracy.

[1]Victim HMD is the detection mechanism under the proposed adversarial attack

C. Process of Crafting the Adversarial Malware

Once the reverse-engineered HMD is built, such as MLP (or any victim defense classifier), the hyperparameters are determined. To launch and craft an adversarial malware, it is non-trivial to determine the level of perturbations that need to be injected into performance counter traces to get the applications misclassified. To determine the number of such HPC events to be generated, we deploy (offline) an adversarial sample predictor. As the ML classifiers are robust to random noises, perturbing the HPC patterns is sophisticated. To perturb HPC patterns, we employ a low-complex gradient loss-based approach, similar to Fast-Gradient Sign Method (FGSM) [23], which is widely employed in image processing.

To craft the adversarial perturbations, we consider the reverse engineered ML classifier with θ as the hyperparameter, x being the input to the model (HPC trace), and y is the output for a given input x, and $L(\theta, x, y)$ be the cost function used to train the reverse-engineered classifier. The perturbation required to misclassify the HPC trace is determined based on the cost function gradient of the chosen classifier. The adversarial perturbation generated based on the gradient loss, similar to the FGSM [14] is given by

$$x^{adv} = x + \epsilon sign(\nabla_x L(\theta, x, y)) \quad (1)$$

where ϵ is a scaling constant ranging between 0.0 to 1.0 is set to be very small such that the variation in x (δx) is undetectable. In the case of FGSM, the input x is perturbed along each dimension in the direction of the gradient by a perturbation magnitude of ϵ. Considering a small ϵ leads to well-disguised adversarial samples that successfully trick the machine learning model. In contrast to the images where the number of features are large, the number of features, i.e., HPCs are limited. Thus the perturbations need to be crafted carefully and also made sure that they can be generated during runtime by the applications.

In contrast to works that assume the application features to be binary, such as [19], this work aims to predict and determine the adversaries for the microarchitectural event patterns, i.e., HPCs, to generate during runtime with the aid of a benign code, which is one of the primary distinctions from existing works. It needs to be noted that determining the required perturbation for a given application is done offline. The process of crafting the adversarial application to generate the perturbations in the HPC trace during runtime is presented next.

V. Adversarial HPC Generation

To generate the required number of HPCs, we craft an application (benign) that wraps the victim application and generates additional performance counter traces that make the overall trace (of the victim application) similar to the predicted HPC trace. We discussed the adversarial HPC predictor previously.

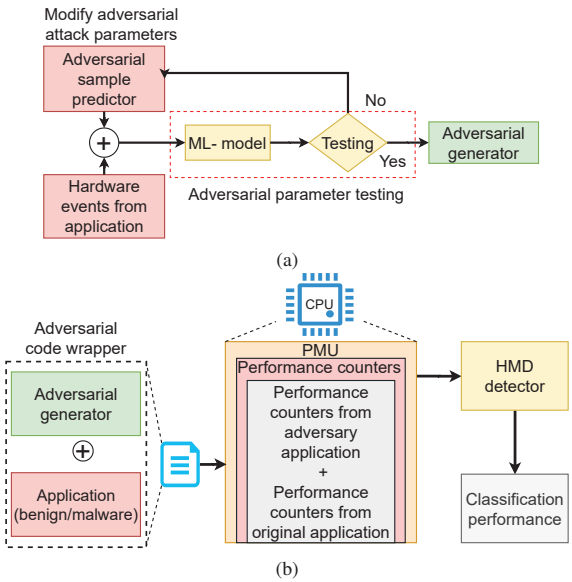

Fig. 3: (a) Determining adversarial code generator parameters with the aid of adversarial HPC predictor; (b) Process of adversarial sample generation with adversarial code to force HMD performance degradation through misclassification of benign/malware applications

We do not know any works in the past that have employed the same approach as our work.

Algorithm 1 Pseudocode for generating adversarial HPCs

Require: Application 'App()'
Ensure: Adversarial microarchitectural events
1: **cache_miss_function()** {Sample pseudo code that generates required number of adversarial LLC misses}
2: #define array[n] % Size of array and loop define amount of variation
3: `load` i #0
4: *Loop 1:* `cmp` i #n {Compare i with n}
5: array[i]=i
6: `jump Loop1`
7: end
8: `load` i #0
9: *Loop 2:* `cmp` i #k {$k <= n$}
10: *ld rax &array[i] # load array address in register rax*
11: *cflush (rax)* {Instruction as a function of array size and loop size}
12: *jump* Loop2
13: end
14: **branch_misses_function()** {Code that generates required number of adversarial branch instructions and branch misses}
15: #define int a, b, c, d
16: a<b<c<d<n
17: *Loop 3:* `cmp` i #a { ··· function ··· }
18: *Loop 4:* `cmp` i #b { ··· function ··· }
19: *Loop 5:* `cmp` i #c { ··· function ··· }
20: *Loop 6:* `cmp` i #d { ··· function ··· }
21: *Loop 7:* `cmp` i #n { ··· function ··· }
22: *jump* Loop 3; end ;
23: {Similar functions to generate other HPCs as predicted by adversarial sample predictor}
24: **APP()** {User/Attacker's application to be executed}

In Algorithm 1, we show the pseudo-code to create adversarial LLC load misses and branch misses. The LLC load misses, and branch misses are some of the pivotal microarchitectural events that malicious applications [2]

To generate LLC load misses, an array of size n is loaded from memory and flushed to generate LLC load misses. This is outlined in Line 2-12 of Algorithm 1. The experiments are repeated multiple times with different array sizes (n) and the different number of elements flushed (k) to determine the number of LLC load misses generated. Further, a linear model is built to find the dependency of n and k on the number of LLC load misses. Once the adversarial sample predictor predicts the number of LLC load misses generated to craft an adversarial sample, the n and k are accordingly determined. We employ a linear model due to its low complexity and high accuracy ($<3\%$ error) to determine the dependency between n and k for our experiments.

Example: For instance, the crafted application similar to that depicted in Line 2-12 of Algorithm 1 with n and k set to 100K leads to an LLC load miss of 73K, whereas when n and k is set to 500K, around 287K LLC load misses are generated. The flushing of the data has been verified by executing attack code with and without flushing the cache lines - the execution time is around $1.5\times$ when the data is flushed compared to the case when data is not flushed.

In a similar manner, branch misses and branch instructions are generated as shown in Line 15-22 of Algorithm 1. To increase the branch misses, a set of conditional statements, i.e., comparison statements, are embedded into the application to create branch misses. The branch instructions depend on the number of conditional statements evaluated. In the presented pseudo code, we have five conditional statements for generating branch-misses (Line 15-22). For the attack code on branch miss events, with a loop size of 20K and integer values assigned to a, b, c, and d, branch misses' value is around 255K. An increase in branch misses is observed by inserting not taken (not executed) dummy loops.

The process flow of adversarial sample predictor is shown in Figure 3(a), where the performance counters from the victim application are combined (offline) with the predicted samples. These are fed to the ML model to gauge its performance. Suppose the ML model classifies the malware and benign with high accuracy. In that case, the adversarial attack parameters are modified; else the predicted samples are utilized for adversarial sample generation during application execution, as shown in Figure 3(b). In Figure 3(b), the overall performance counters seen by the system are a result of the original application and the adversarial code. The HMD profiles the applications in runtime. If the predicted HPC values are smaller than those generated by original applications, we insert delay elements to smoothen the HPC trace and reduce the HPC values. It needs to be noted that using this process we generate adversaries to force the HMD to misclassify benign as malware and malware as benign applications.

VI. Hardening HMD against Adversarial Malware

We have discussed how attacks are performed to trick the HMD and force misclassification. The adversarial malware is crafted to perturb the HPC patterns and hence trick the victim HMD. Although HMD-Hardener can be employed for different defense strategies to ensure hardening for best performance

under different adversarial attack types, we keep the discussion limited to 'FGSM' type attack and 'Adversarial Training' (FGSM in this work) type defense for conciseness. Adversarial training is one of the initial solutions introduced as a way for ML classifiers and deep learning classifiers to battle against adversarial samples. The method of adversarial training focuses on having adversarial samples used to train the model/classifier. They obtain the adversarial information in the training stage itself and stay robust against such attacks. We retrained the HMD using adversarial samples, and it is observed to deliver robust performance under an adversarial attack. The assumption is that we know the type of attack that can happen and the attack parameters.

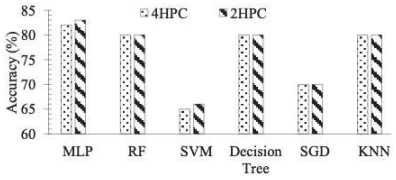

Fig. 4: Accuracy results for various ML classifiers with feature size (HPCs) of two and four

VII. Results and Evaluation

In this section, we present the accuracy of HMD in classifying malware and benign applications. Further, we give the impact of an adversarial malware attack on the HMD and attack resiliency post hardening. Finally, we present hardware implementation results for the ML classifiers.

A. Experimental Setup and Data Collection

The applications (both malware and benign) are executed on an Intel Xeon X5550 machine running Ubuntu 18.04. We execute more than 3000 benign and malware applications for HPC data collection. Benign applications include MiBench benchmark suite [24], Linux system programs, browsers, text editors, and word processor. For malware applications, Linux malware is collected from virustotal.com [25] and virusshare.com [26]. Malware applications include five classes of malware comprising of 607 Backdoor, 532 Rootkit, 2739 Virus, 1264 Worm and 7221 Trojan samples. All the applications (malign/benign) are profiled in Linux Containers. The adversarial sample predictor is implemented in Python using the Cleverhans library. The linear model is derived using the traditional statistical curve fitting technique. The adversarial sample generator is implemented using C and executed on a Linux terminal.

B. HMD Classification Accuracy

Figure 4 shows a comprehensive accuracy comparison of various ML classifiers used for malware detection. We implemented six general ML classifiers. The accuracy of malware detection with two feature sizes (4 and 2) are reported. As seen in Figure 4, MLP, random forest (RF) and decision tree classifiers perform very well for both the 4HPC and 2HPC as feature sizes. High performance with fewer features enables the HMD-Hardener to classify applications and detect malware in runtime with less overhead on the system, making repeated calls to the *Perf* tool. For instance, as shown, MLP achieves close to 82% accuracy, 80% for random forest (RF), 79% for SVM, and so on with four HPCs. However, we observe that reducing the number of vital performance counters to 2 results in similar classifiers' accuracy. We also observe that the HMD achieves detection accuracy in the range of 84-90% with 16 and 8 features. The higher gain results in overhead; results not shown for conciseness. For this work, we will consider the accuracy with 4HPCs with which the classifiers such as MLP, RF, decision tree, and KNN perform well with around 82% detection accuracy on an average. Four HPCs are easily possible to be captured in runtime to allow malware detection. We also evaluated the aforementioned classifiers' performance by observing the Precision, F1-Score, and Recall metrics. These metrics' values are approximately similar to the accuracy metric's values shown in Figure 4.

TABLE II: Impact of adversarial attack on HMD

	Accuracy	Precision	F1-score	Recall
Before attack	82%	78.1%	78.1%	82.1%
After attack	18.1%	45.0%	10.0%	18.0%
After hardening	81.2%	80.1%	80.1%	81.2%

TABLE III: Post synthesis hardware results of different ML classifiers (@100MHz) when deployed in HMD-Hardener

Classifier	Power (mW)	Energy (mJ)	Area (mm^2)
MLP	90.45	5.12	4.5
RF	40.64	2.35	2.25
SVM	45.63	2.79	1.81
Decision Tree	36.54	2.29	1.55
SGD	54.46	3.21	1.46
KNN	44.81	3.37	1.27

C. Impact of Adversarial Attack

We depict the impact of adversarial sample generator on the performance counters in Figure 5 that shows the LLC load misses for a benign application (ISCAS'85). The adversarial pattern predicted by the adversarial sample predictor is shown in Figure 5(a). We observe that there exist some spikes in the pattern as marked in the figure. Figure 5(b) shows the HPC pattern generated when the application is integrated (wrapped) with the adversarial HPC generator. On average, there is an error of 2.2% between the trace predicted by the adversarial sample predictor and the trace generated by the adversarial sample generator. This indicates the adversarial generator can efficiently generate the required number of perturbations in the HPC traces. The MLP classifier delivers 82% accuracy on average in detecting malware. Post adversarial attack, the accuracy of the MLP drops to 18.1%, indicating that the adversarial sample generator degrades the HMD performance. The results of the MLP classifier's accuracy, precision, F1, and recall metrics are presented in Table II. We observe similar results, shown in Figure 5, for branch miss type adversarial perturbations.

D. Adversarial Learning - Hardening

For hardening the HMD, it needs to be trained on normal samples and adversarial samples as well. For MLP classifier, the accuracy is restored close to the original accuracy before the attack, as presented in Table II. We observe similar results with other classifiers after hardening, thus verifying that the HMD can become resilient against an adversarial attack, provided it is trained on adversarial samples. The HMD is trained on a

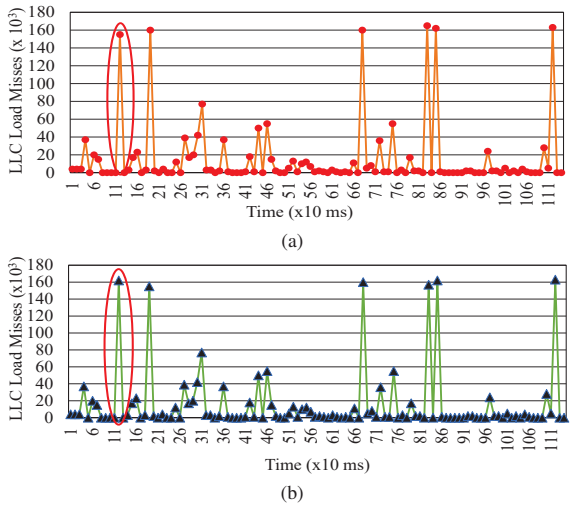

Fig. 5: (a)LLC load miss trace of the application predicted by adversarial sample predictor; (b) Generation of LLC load miss trace by adversarial sample generator

new dataset, containing the original and adversarial samples combined. As the HMD is robust against adversarial samples, it delivers high performance in detecting the malware and benign samples, as shown in Table II. Hence, if the HMD deployed in a system is adversarially trained on the perturbed HPC traces generated by using the process discussed thus far, the HMD is hardened against adversarial malware samples. This ensures the HMD delivers high performance against the normal and adversarial attack samples in runtime.

E. ASIC Implementation of Classifiers in HMD-Hardener

We conduct comprehensive hardware implementation of the classifiers embedded into HMD on ASIC. All the experiments are implemented on a Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit, 28 nm SoC running at 1.5 GHz. The power, area, and energy values are reported at 100MHz. We used Design Compiler Graphical by Synopsys to obtain the area for the models. Power consumption is obtained using Synopsys Primetime PX. The post-layout area, power, and energy are summarized in Table III. Among all the classifiers, MLP consumes highest power, energy and area on-chip (Table III). The post-layout energy numbers were almost 2x higher than the post-synthesis results. This increase in energy is mainly because of metal routing resulting in layout parasitics. As the tool uses different routing optimizations, the power, area, and energy values keep changing with the classifiers' composition and architecture.

VIII. Conclusion

In this work, we propose an adversarial attack on microarchitectural event-based malware detection systems (HMD). The HMD systems are utilized for detecting and classifying malware. This work employs an adversarial sample predictor to determine the HPC count required to degrade HMD performance. Post determining the required number of HPC count, using the proposed adversarial sample generator, the required HPC trace is generated without intervening with the original application and eventually leading to misclassification. Furthermore, the malware detection accuracy is reduced from 82% to 18.1%. To make HMDs resilient to adversarial attacks, we proposed training with the adversarial samples (HMD hardening), promisingly boosting the HMD performance close to the original performance before attack to 81.2%. The hardware implementation results are presented to demonstrate latency and area overheads per ML model to select a more suited model for runtime detection.

References

[1] G. Jacob *et al.*, "Behavioral detection of malware: from a survey towards an established taxonomy," *Journal in Computer Virology*, vol. 4, no. 3, pp. 251–266, Aug 2008.

[2] N. Patel *et al.*, "Analyzing hardware based malware detectors," in *Design Automation Conf.*, 2017.

[3] J. Demme and et al., "On the feasibility of online malware detection with performance counters," *SIGARCH Comput. Archit. News*, vol. 41, no. 3, pp. 559–570, Jun 2013.

[4] A. Tang *et al.*, "Unsupervised anomaly-based malware detection using hardware features," in *Research in Attacks, Intrusions and Defenses*, 2014.

[5] H. Sayadi and et al., "Ensemble learning for effective run-time hardware-based malware detection: A comprehensive analysis and classification," in *Design Automation Conference*, 2018.

[6] F. Brasser and et al., "Advances and throwbacks in hardware-assisted security: Special session," in *Int. Conf. on CASES*, 2018.

[7] S. Dinakarrao and et al., "Lightweight node-level malware detection and network-level malware confinement in iot networks," in *Design Automation and Test Con. in Europe*, 2019.

[8] H. Sayadi and et al., "2SMaRT: A two-stage machine learning-based approach for run-time specialized hardware-assisted malware detection," in *Design Automation and Test Con. in Europe*, 2019.

[9] X. Wang *et al.*, "Malicious firmware detection with hardware performance counters," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 2, no. 3, pp. 160–173, 2016.

[10] S. M. P. Dinakarrao *et al.*, "Cognitive and scalable technique for securing iot networks against malware epidemics," *IEEE Access*, vol. 8, pp. 138 508–138 528, 2020.

[11] A. Garcia-Serrano, "Anomaly detection for malware identification using hardware performance counters," *CoRR*, vol. abs/1508.07482, 2015.

[12] K. Khasawneh and et al., "EnsembleHMD: Accurate hardware malware detectors with specialized ensemble classifiers," *IEEE Trans. on Dependable and Secure Computing*, 2018.

[13] C. Szegedy and et al., "Intriguing properties of neural networks," in *Int. Conf. on Learning Representations*, 2014.

[14] I. Goodfellow *et al.*, "Explaining and harnessing adversarial examples," in *International Conference on Learning Representations*, 2015.

[15] N. Papernot and et al., "The limitations of deep learning in adversarial settings," in *IEEE European Symp. on Security and Privacy*, 2016.

[16] Y. Liu *et al.*, "Delving into transferable adversarial examples and black-box attacks," in *Int. Conf. on Learning Representations*, 2017.

[17] S. M. P. Dinakarrao *et al.*, "Adversarial attack on microarchitectural events based malware detectors," in *Design Automation Conference*, 2019.

[18] A. P. Kuruvila *et al.*, "Defending hardware-based malware detectors against adversarial attacks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.

[19] A. Huang and et al., "Adversarial deep learning for robust detection of binary encoded malware," *CoRR*, vol. abs/1801.02950, 2018.

[20] K. Khasawneh and et al., "RHMD: Evasion-resilient hardware malware detectors," in *IEEE/ACM Int. Symp. on Microarchitecture*, 2017.

[21] B. Zhou and et al., "Hardware performance counters can detect malware: Myth or fact?" in *ASIACCS*, 2018.

[22] A. Kurakin *et al.*, "Adversarial examples in the physical world," *CoRR*, 2016.

[23] I. J. Goodfellow *et al.*, "Explaining and harnessing adversarial examples," 2014.

[24] M. R. Guthaus and et al., "MiBench: A free, commercially representative embedded benchmark suite," in *IEEE Int. W. on Workload Characterization*, 2001.

[25] (2020) Virustotal intelligence service. Last accessed: 05-Dec-2020. [Online]. Available: www.virustotal.com/intelligence

[26] (2020) Virusshare team. Last accessed: 05-Dec-2020. [Online]. Available: www.virusshare.com