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.
2021 Design, Automation & Test in Europe Conference & Exhibition.

Problem Addressed in the paper :
Software-Based Malware detection techniques has too much performance overhead and computational compexity for the system as the malware developers develops more sophisticated malware that are hard to detect. To overcome this problem Hardware-Based Malware detection (HMD) technique emerged as a panacea to detect malicious application and enhance security of the system. HMD utilizes low-level microarchitectural hardware events for detecting and classifying the malware from benign application. In this paper they propose an adversarial attack on HMD in which the adversarial sample are generated through benign code that is wrapped around a benign or malware application to misclassify the application. Attacker has no direct access to modify the HPC and manipulation of HPC is highly complex to perform code obfuscation for executing malware. First, we assume the victim's defense system is unknown and perform reverse engineering to mimic the embedded HMD's behaviour and build Machine learning classifier. To determine the required number of HPCs to be generated through the application to be missclassified, they employ an 'adversarial sample predictor' which predicts the number of HPC to be generated to misclassify an application by the HMD. To add the required HPCs to an application, they craft an 'adversarial HPC generator' application that generate the required number of HPCs. This adversarial HPC generator application is wrapped around the application that needs to be misclassified by perturbation. The proposed work benefits from the following.

- No need to tamper or modify the source code of the application around which the proposed adversarial sample generator code is wrapped.
- The crafted application has no malicious features embedded, thus not detectable by malware detectors.
- Scalable and flexible, the crafted application can generate events required to create a powerful adversary.

To make the HMD robust and resilient to such adversarial attacks, they 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.

- Craft an adversarial attack on HMD.
- Novel way of crafting adversarial HPC traces through a bening application.
- To generate false alarms to weaken the trust on HMD.
- Make HMD robust and resilient to adversarial attacks.

Motivation for the Problem :
HMD is best suited to be the replacement of a software-based malware detection as it have low performace overhead. We need to figure out that is the HMD really the best in it's buisness before adapting it to detect malware. So in this paper they try to exploit the HMD so that the trus on HMD can be weaken and also overcome the problems related with HMD.

Motivation for the Solution :
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 behaviour. It also perform a adversarial learning by training the HMD on adversarial samples.

Theory :
HMD.

Initial step in training process is to generate dataset that can be used to train machine learning models. The dataset captured features that describe the hardware's microarchitectural state at different time instances for application executing on the system. We capture the HPC values of application running on the system using perf tool (available in linux). Running application generates a very high dimension dataset. Instead of considering all captured features, irrelevant data is identified and removed using a feature reduciton algorithm. They collected 44 performance counters, then find the most correlated features using the Correlation Attribute Evaluation. Next, the features are scored based on their importance and relevance to the target variable through the feature scoring process. The HMD delivers high performance in detecting malware with four features, and collecting four features in runtime generates less overhead compared to eight. The input variable to HMD classifiers are the HPC that are extracted every 10ms interval from the running application, and the output variable is the class of an application.

Adversarial attacks on ML-Classifiers.

To mimic the functionality of victim HMD detector, we perform reverse engineering. At first, create a dataset that contain both benign and malware application. The victim HMD is fed with all the application and their response has been recorded. These response are utilized to train the different ML classifiers to mimic the functionality of victim HMD. Trained ML classifiers response is compared with the output of victims HMD. To ensure reverse engineering is performed efficiently, we train multiple ML classifiers and choose the classifier that yields high performance. All the above process is shown in below figure.

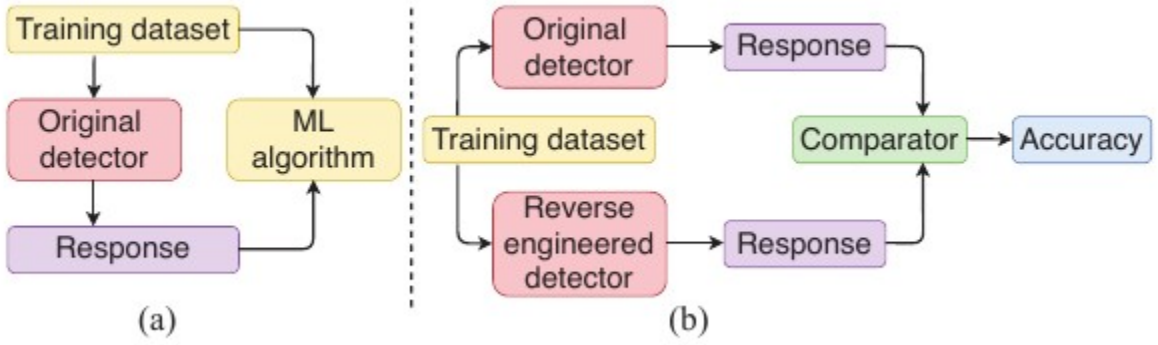

Figure 1: (a) process of reverse engineering an HMD; (b) Testing performance of reverse-engineered detector.

Now we need to find the level of perturbations that need to be injected into performance counter traces to get the application misclassified. Finding it is non-trivial. It needs to be noted that determining the prequired perturbation for a given application is done offline. The used an adversarial sample predictor to determine the number of HPC event to be generated. The perturbation required to misclassify the HPC trace is determined based on the cost function gradient of chosen classifier.

$$X^{adv} = x + \epsilon sign(\delta_x L(\theta,x,y))$$

To generate the required number of HPCs, we craft an applicaiton that wraps the victim application and generates additional performance counter traces that make the overall trace similar to the predicted HPC trace. The LLC load misses and branch misses are some of the pivotal microarchitectural events that malicious applicaiton have. So they pseudo-code to create LLC load misses and branch misses.

Algorithm 1 Pseudocode for generating adversarial HPCs

Require: Application 'App()'
Ensure: Adversarial microarchitectural events

```
cache_miss_function() {Sample pseudo code that generates required number of adversarial LLC misses}
    #define array[n] % Size of array and loop define amount of variation
        load i #0
    Loop 1: cmp i #n {Compare i with n}
        array[i]=i
        jump Loop1
    end
    load i #0
    Loop 2: cmp i #k {k <= n}
        ld rax &array[i] # load array address in register rax
        cflush (rax) {Instruction as a function of array size and loop size}
        jump Loop2
    end
branch_misses_function() {Code that generates required number of adversarial branch instructions and branch misses}
    #define int a, b, c, d
    a<b<c<d<n
    Loop 3: cmp i #a { ··· function ··· }
    Loop 4: cmp i #b { ··· function ··· }
    Loop 5: cmp i #c { ··· function ··· }
    Loop 6: cmp i #d { ··· function ··· }
    Loop 7: cmp i #n { ··· function ··· }
    jump Loop 3; end ;
{Similar functions to generate other HPCs as predicted by adversarial sample predictor}
APP() {User/Attacker's application to be executed}
```

To generate LLC load misses, an array of size n is loaded from memory and flushed to generate LLC load misses. The experiment are repeated multiple times with different array size (n) and the different number of elements flushed (k) to determine the number of LLC load misses generated. 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. In a similar manner, branch misses and branch instruciton are generated.

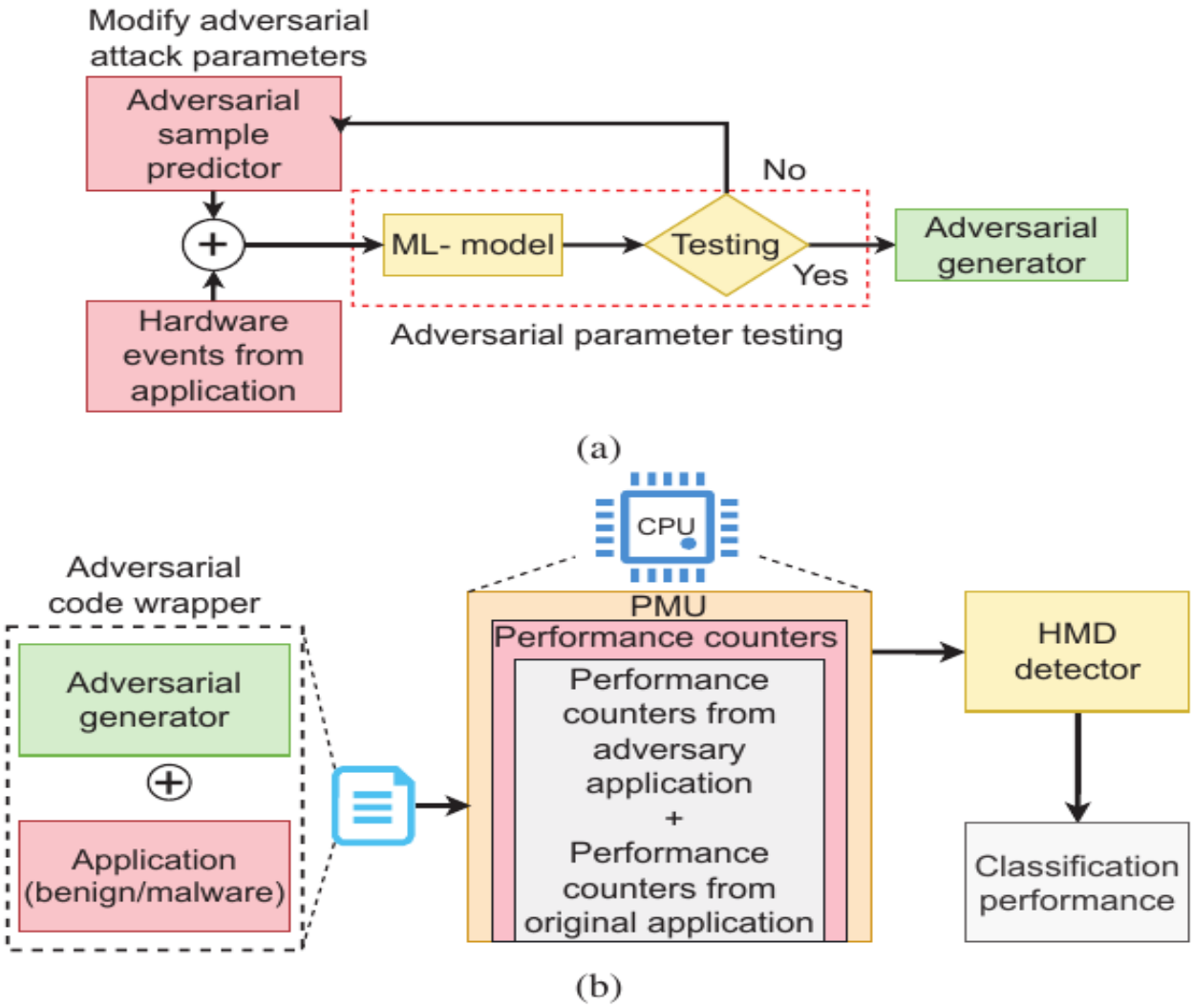

Figure 2: (a) Determining adversarial code generator parameters with the aid of adversarial HPC predictor; (b) Process of adversarial sample generation with adversarial code to force HMD performance degradation through misclassification of benign/malware applications.

In Figure 2(a), the performance counters from the victim application are combined with the predicted samples. These are fed to the ML model to gauge its performance. If ML model classifies the malware and benign with high accuracy then the adversarial attack parameters are modified else the predicted samples are utilized for adversarial sample generation during application execution. In figure 3(b), the

overall performance counters seen by the system are a result of the original application and the adversarial code.

Hardening HMD against Adversarial Malware.
The HMD is tricked by adversarial malware by crafting the perturbation in HPC patterns. We can employ different defense strategy to ensure hardening of HMD for best performance under adversary attack. Adversarial training is one of the initial solutions introduced to hardened the HMD. This method focus on training the ML classifier with the adversarial samples. So, they obtain the adversarial information in the training stage itself and stay robust against such attacks.

Implementation :
The application are executed on an Intel Xeon X5550 machine running Ubuntu 18.04. They execute 3000 benign and malware appliation for HPC data colleciton. Malware application is collected from virustotal.com and virusshare.com. The adversarial sample predictor is implemented in python using the *cleverhans library.* The linear model (n and k) is derived using the traditional statistical curve fitting techinque. The adversarial sample generator is implemented using C and executed on a Linux terminal.

Critique :
They propose adversarial training for the HMD hardening so that HMD can detect adversarial malware. What if the malware developer again reverse engineer the HMD and follows the same process that discuss above to create adversarial sample that tricked the adversarial trained HMD to misclassify an application.
If we have malware and we want it to bypass by detector. Adversarial sample predictor will predict the number of required HPC to be generated. But malware does not required more HPC it need to reduce the HPC values so that it can bypass HMD. Proposed solution does not comment anything about that.
We can add some benign code to malware application but here the benign code will not generate any HPC values. The benign code is executed in every observation window along with malware so that HPC value decrease.

Related works :
Some works have shown that by deploying Machine Learning techniques fed with low-level microarchitectural events captured by hardware performance counters can aid in differentiating benign applications. “*On the feasibility of online malware detection with performance counters*” works first propsed the use of performance counter for the malware detection. Afterwards various ML classifiers are tested and compared their performance to analyse the best classifier for the detection of malware. Some works exposed the vulnerabities of ML classifiers result in changing the outcome of the classifier by adding specially crafted perturbations to the input data.