Defending Hardware-based Malware Detectors against adverserial Attacks

Abraham Peedikayil Kuruvila, Student Member, IEEE, Shamik Kundu, Student Member, IEEE,

Kanad Basu, Member, IEEE

Published 7 May 2020

Problem Adressed :

Malicious software are designed by attackers in order to harm a system or leak sensitive information significant to the user. In order to prevent this kind of attack, Hardware Malware Detection (HMD) technique is implemented at hardware level which is very efficient and have low hardware overhead. HMD uses various Machine Learning classifier to classify whether the running application is benign or malicious. Hardware program counter (HPC) are special registor build into all modern processor that count significant low-level microarchitectural feature like branch misses, cache misses and CPU cycles etc. These HPC values are harvested by HMD and used to train the ML classifier in Malware detector. Recently, adversarial attacks are being developed on HPC based malware detectors. In one such attack, perturbations are introduced into the HPC traces with an aim to subdue the HMD. In this attack, the attacker considers the victim's defense system to be a black box and then reverse engineers the HMD so as to replicate the behaviour of the security system. This reverse engineering process furnishes an optimal machine learning classifier that utilizes the embedded feature set of harvested HPCs from the application data. The attacker then employs an adversarial sample predictor to predict the number of HPCs to be generated to impel the Malware detector into misclassifying an application. Following this, an adversarial sample generator is developed to execute in conjunction with the target application which generates the preset HPC values acquired from the adversarial sample predictor. These fabricated values engendered from the adversarial attack result in misclassification. To restore the trust on HMD which is eroded by the adversarial attack, this paper counters this attack by proposing a Moving Target Defense (MTD) that utilizes various ML classifiers deployed dynamically.

Motivation for the Problem :

There are many malicious software are there which can harm your system or leak sensitive information that one don't want to share. We see many such attack in recent times and not every user is well aware of cause and effect of such types of attacks. Therefore, we need a mechanism which can detect the malicious application in our system and prevent them from harming our system. HMD is effective in detection of such attack untill an adversarial attack erode the trust of HMD detection. So, we need a mechanism on which an adversarial attack is ineffictive and we get our trust back on HMD.

Motivation for the Solution :

In adversarial attack, attacker mimics the victims HMD by Reverse Engineering it and add perturbations to the input traces so that it misclassify an application. If the attack surface available to the attacker is continously altered by modifying some of parameter employed in malware detection then it is almost impossible to mimics the victim HMD. MTD is one such promising cyber-defense strategy. It constantly drives attack surface to evolve across multiple system dimensions. Reducing or changing the attack surface dynamically confuses the planned attack.

Theory :

During an adversarial attack, only specific HPCs are being targeted by an attacker. Therefore, a defense that constantly changes the attack surface by modifying the HPC combinations possible is robust in countering this adversarial attack. The proposed algorithm perpetually modifies the following parameters employed in malware deteciton.

1. the count of HPCs harvested from the processor.
2. the specific set of HPCs to be provided as input to the HMD.
3. which classifier of the HMD is utilising which HPC combinations.

With a plethora of possible combination for generating ML classifiers with unique HPC clusters, the best HPCs need to be utilized in order to obtain the highest accuracy. Feature Testing is used to address this challenge. There are three different features testing techniques employed to assist in producing ML models with good accuracy. *Univariate Selection, Feature Importance and Correlation Matrix with Heat map*. These three techniques applied in building the MTD. An analysis can be preformed to determine the best HPC combinations for training a classifier from the results of the three techniques. Since the goal of our proposed MTD is to create multiple ML classifiers while maintaining acceptable accuracy, the best HPCs should be paired with HPCs that are average to ample.

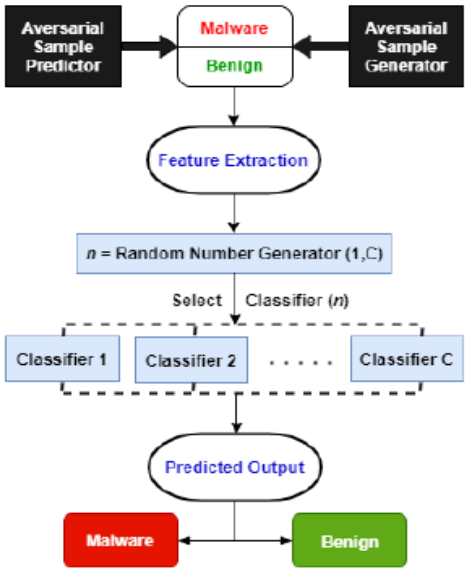

Figure 1: A process flow diagram for proposed MTD.

Once the best and optimal number of classifiers and HPCs are selected, the MTD can be modelled. HPC data are collected in specific intervals (10ms). Each interval act as iteration. For each iteration, a random number will be generated between a range of (1,N) where N is the number of classifiers being utilized in the MTD. The random value will determine which classifier is used to predict whether the given HPC iteration data corresponds to benign or malicious.

The minimum number of HPCs corresponding to each classifier used in HMDs is 1, and maximum number of HPCs that the CPU allows to observe simultaneously. We can calculate the possible classifier combination that the model can obtain by given the number of total HPCs available and the maximum number of HPCs that a ML classifier can accomodate. Let H_t be the total number of HPCs in the processor and X be the number of counters that is used for a ML classifier. R_{max} be the number of HPCs that can be monitored simultaneously for a particular processor. X ranges from 1 to $R_{max.}$ For a Raspberry pi 3 with ARM Cortex-A53 CPU, the value of H_t is 20 and R_{max} is 4. So the possilbe classifiers models with distinct set of counters for a processor is 6195. A defender can select any number of classifier from this set for the MTD model. Let C be the number of classifiers that is selected from the set of classifiers (say N_h). The value of C constrained to $2 \leq C \leq N_h$. For a raspberry pi 3 with 20 HPC have approximately 7.6×10^{1864} cobinations that the defender can use to model a MTD. An adversary attempting to break into the system has to know exactly what HPCs a particular classifier is using which is having a probability of $1/7.6 \times 10^{1864}$ for even a small processor with 20 HPCs. Let us consider a situation, where the malware detector employs a single classifier with a fixed number of HPCs(h) for classifying an applicaiton. Now with *h* HPCs amont H_t, an attacker can chosse between $\binom{Ht}{h}$ combinations of HPCs. Among these only one combination is correct, hence the probabiliy of success is very low.

Implementation :

They conduction test on a *Raspberry pi 3 model B,* which utilizes an ARM processor, and HPC values are collected using perf tool. The ML models utilized in the MTD were Descision Tree, Neural Network classifiers. These models were written in python using scikit-learn. The malware samples were collected from Virusshare and the benign programs include the MiBench benchmark as well as different sorting and computing algorithms. 300 Malware and 300 benign programs were executed on the Raspberry Pi, and the HPC values were harvested. For each HMD, a classifier was trained on HPC branch-instrucitons, HPC branch-misses, HPC instructions, and HPC LLC-load-misses. The adversarial attack for branch misses and LLC load misses were written as C code, which were then ran together with the programs to be camouflaged through a python wrapper. 50 new malware samples, which were not included in the training dataset, were tested on each trained classifier to prove that the accuracy would go down. These 50 were first tested on each ML classifier with no HPC perturbations and then these malware samples were executed again with the adversarial HPC generator and each classifier was tested to demonstrate reduction in accuracy. Then finally, these malware samples with HPC perturbations were executed through the proposed MTD to prove that the defense mechanism is effective in recovering the lost accuracy. In the MTD, two classifier using the same ML algorithm were implemented. One of classifier using HPCs Branch-instruction, Branch-misses, Bus-cycles, Cache-misses and other classifier using Cache-references, CPU cycles, and Instrucitons for the other classifier.

Critique :

The main idea is to add some randomness to the HMD so that it is almost impossible to reverse-engineering the victim HMD. Even if they uses only two ML classifiers or use one classifier against the possible combinaiton of HPCs then the probability of choosing the correct combination is too low. This shows the effectiveness of MTD in HMD and restore the trust in HMD. The proposed solution worked well against advarsarial attack and attacker will unable to reverse-engineer the detector. This also restore the accuracy of HMD which was degraded in advarsarial attack.

We can also considered the sceneria where the window size used to collect the classification features varies. We can prepare multiple pairs of attacker testing and training datasets, using different collection of periods. The accuracy of model will match when windows size for both victim and attacker matches. But by changing the instruction window make it unable to reverse-engineer the victim HMD.

Regarding Related Work :

As the author wants to overcome the failure of HMD because of an adversarial attack. So they selected the papers that shows the HMD mechanism in detection of malware and benign application and also choose the paper in which an adversarial attack is proposing agains HMD. The author also picked the papers in which possible solution to adversarial attack proposes with low hardware cost. MTD is a deffense mechanism used against the attackers in which they continously changing the attack surface. So the author reads some of papers that shows the working principal of MTD.

List of impactfull related works.

-> 3 reasons why moving target defense must be a priority.

-> On the general applicability of instruction-set randomization.

-> Rhmd evasion-resilient hardware malware detectors.

-> Lightweight node-level malware detection and network-level malware confinement in iot networks