Monotonic-HMDs: Exploiting Monotonic Features to Defend Against Evasive Malware

Md Shohidul Islam*§, Behnam Omidi*, Khaled N. Khasawneh*
*ECE Dept., George Mason University, email: {mislam20, bomidi, kkhasawn}@gmu.edu
§CSE Dept., Dhaka University of Engineering & Technology, Gazipur

Abstract—**Machine learning-based hardware malware detectors (HMDs) offer a potential game-changing advantage in defending systems against malware. However, HMDs suffer from adversarial attacks; they can be effectively reverse-engineered and subsequently be evaded, allowing malware to hide from detection. Adversarial evasion attacks requires adding benign features to the program execution to be able to evade detection. Against these attacks, in this paper, we propose Monotonic-HMDs, which are HMDs built using monotonic features to defend against adversarial evasion attacks. Specifically, Monotonic-HMDs are build using monotonic malicious features only. Thus, Monotonic-HMDs ensures that an adversary cannot evade the detection by simply adding benign features to the malware programs since they are not used in the Monotonic-HMD model. In addition, adding malicious features will only increase the probability of detecting the input program as malware. Our experimental results demonstrate that Monotonic-HMDs offer effective defense against adversarial attacks without sacrificing significant detection accuracy, which can be interpreted as a cost for security in classifying malware. Importantly, our results shows that for evasive malware that can completely evade current HMDs, the proposed Monotonic-HMDs achieve 83% detection accuracy and maintain this accuracy even under more aggressive attacks. Moreover, Monotonic-HMDs reduce the inference time, i.e., time to perform one detection, by 61.11%. Furthermore, the hardware implementation results of the Monotonic-HMDs shows that Monotonic-HMDs offers area and power consumption savings compared to current HMDs.**

Index Terms—**Adversarial machine learning, monotonic models, malware detection**

I. Introduction

Malware attacks are among the growing security threats to modern computing systems. While manufacturers are continuously making efforts in building secure systems, the number of system vulnerabilities are still overwhelming. Sophisticated attackers exploit these vulnerabilities and develop numerous types of malware to fulfill their malicious goals [15], [19]. While preventing malware from compromising the system is impossible, protecting systems from malware attacks requires detection, which is mainly accomplished using two approaches: static and dynamic detection. Static approach basically detects malware by matching their signatures with previously stored database; however, this technique becomes ineffective in detecting obfuscated/metamorphic/polymorphic malware and zero-day attacks whose signatures have not yet been encountered [12]. Overcoming these challenges leads to the concept of dynamic detection, which does not rely on malware signature. Instead, dynamic approach detects malware by monitoring their run-time behaviour, which makes the technique robust in detecting unseen signatures []. However, dynamic detection, when done with software implementation, poses significant overhead on system performance and power.

To make the continuous dynamic detection more efficient, a new implementation technique has emerged called Hardware Malware Detectors (HMDs) [1], [6], [9], [10], [13], [14], [17], [18]. HMDs are essentially machine learning classifiers that detect malware as computational anomaly by using lower-level hardware features. Research shows that HMDs can be built using various hardware features including executed instruction traces, memory access patterns, or other architectural events (e.g., cache miss rates, branch miss rates, etc) collected through Hardware Performance Counters (HPC), which are available in all modern computers. HMDs receive popularity in industries too; the SnapDragon processor of Qualcomm appears to be using hardware features to detect malware attacks; however, the technical details are not published [16]. HMDs offer great advantage as they can detect malware by staying always 'on' with small-to-no impact on system performance and power [14], [15].

However, attackers are always in pursuit of gaining upper hand by adapting malware so that they can bypass malware detectors. As a results several works [2], [4], [7] showed that it is indeed possible to craft evasive malware that can evade HMDs' detection while continuing their intended malicious activities. These evasive malware attacks assume black-box access to the HMDs, i.e., attackers do not know the victim HMDs model internals (e.g., model architecture, hyper-parameters, weight, bias, etc). But attackers can query the victim HMDs by supplying inputs and observing model outputs. Due to unavailability of victim model parameters, black-box approach takes two steps: (1) reverse engineering the victim HMDs, which builds (trains) proxy models, (2) generate evasive malware based on proxy models. There exists another threat model called white-box attack, which assume that attackers have access to victim HMDs model parameters, and thus attackers follow only the second step – generating evasive malware based on the exact victim HMDs model.

To defend HMDs from evasive malware attacks it was shown that retraining HMDs with evasive malware samples is limited [7], i.e., it is still possible to evade the re-trained HMD. Furthermore, randomization based defenses, called evasion-resilient hardware malware detectors (RHMDs) [7], were proven to be effective against black-box attacks. Specifically, RHMDs is an HMD organization that uses multiple diverse detectors and randomly switches between them at inference time, i.e., run-time. Such random switching of the base detectors at inference time leads to unpredictable decision boundaries, which makes the reverse engineering harder in case of black-box attack scenario. Consequently, attackers have hard time in building proxy models and generating evasive malware samples. Nonetheless, RHMDs are found to be vulnerable to white-box attacks as the attackers have exact knowledge of the victim HDMs models and can exploit them in generating evasive malware samples [7]. Moreover, building multiple base detectors in RHMDs requires huge dataset and diverse features, leading to increased complexity in their hardware implementation.

In this paper, we introduce Monotonic-HMDs to harden HMDs against evasive malware. Our approach is based on the observation that adversaries target the benign features, i.e., features that contribute to increase the probability of detecting the input as benign, by adding more of them to craft evasive samples of the malware that can evade detection. Therefore, Monotonic-HMDs models use malicious features only for detection, i.e., features that contribute to increase the probability of detecting the input as malicious/malware. As a result, adding or subtracting benign features from the malware will have no effect on the detection since the benign features are not used for detection. Furthermore, Monotonic-HMDs not only uses malicious features for detection, but also imposes monotonic constraints on them so that the HMDs' detection is an increasing function of the malicious features value. As a result, adding more malicious features to the malware, will only results in increasing the probability of the input being a malware. This is a strong security advantage since an adversary who can only increase the feature values (but not decrease them) cannot make perturbations to the malware that fool the HMD into detecting malware as benign programs. As a consequence, to fool the Monotonic-HMDs, an adversary must reduce the number of malicious features in the malware, which would effect the malicious functionality of the malware. For example, cache side-channel attacks requires continuously accessing the cache to be able to extract sensitive information from the victim and reducing the frequency of the cache access reduce the success rate of the attack [5], [21].

While Monotonic-HMDs are not a perfect defense against evasion, we hope that Monotonic-HMDs raises the bar. In particular, since evading Monotonic-HMDs, even under the white-box attack scenario, can't be done through simple evasion automated techniques [2], [4], [7] and requires advanced domain expertise and non-trivial efforts from the adversary.

The key contributions of this paper are as follows:

- We propose Monotonic-HMDs as a defence against evasive malware attacks. Our security analysis shows that Monotonic-HMDs are robust against against both black-box and white-box attacks scenarios.
- We evaluate the Monotonic-HMDs model overhead, in terms of inference performance and model size. Our case study results shows that Monotonic-HMDs has 2.5X faster inference and 12.01% memory space saving compared to regular HMDs.
- We evaluate the hardware complexity of integrating Monotonic-HMDs into an open core. Our evaluation shows that Monotonic-HMDs require significantly lower hardware resources compared to regular HMDs and reduce power consumption.

II. Data set & features collection

This section explains how we collected features and built our dataset comprising benign programs and malware programs. We executed 600 benign programs of numerous types such as text editing tools, browser programs, system programs, SPEC 2006 benchmarks, notepad++, Acrobat Reader, Winrar, etc. As for malware programs, we downloaded 3000 malware samples of different types, including Password Stealers (PWS), Backdoor, Trojan, Worm, Rogue, from a malware database that is called MalwareDB [22]. The malware as well as the benign programs were executed on a 32-bit Windows 7 virtual machine. Note that in order to allow the malware to run freely and perform their intended malicious activities we disabled Windows security and Firewall services. While running benign programs, we interacted with them manually to get a representative results. In contrast, we observed that malware programs did not require user interaction in order to perform their malicious activities. To collect dynamic profiling information and low-level hardware features of a running program, we used Intel Pin instrumentation tool [11]. Dynamic traces of a running program were collected for a duration of 5000 system calls or 15 million committed instructions, after the warm-up period. The collected features are based on the executed instructions frequency, similar to [7].

Finally, we divided the dataset into *victim training* (50%), *attacker training* (25%), and *testing* (25%). Here, *victim training* refers to the dataset that was used to train the victim HMDs, *attacker training* dataset was used to train proxy models for reverse engieering, and *testing* dataset was used for inference and measuring the effectiveness of reverse engineering. In order to avoid bias in the dataset, we distributed different malware types evenly across these datasets.

III. Adversarial attacks on HMDs: The threat model

In this section, we describe in details our baseline HMD (i.e., victim HMD) and the threat model considered. In particular, this paper assumes two attack scenarios (black-box and white-box attack) and we will show that the current implementation of HMDs is vulnerable to both attack scenarios.

A. Victim HMD

A typical victim HMD is machine learning classifier built with low-level hardware features in order to classify malware and benign programs. We train a victim HMD using a logistic regression classifier with low-level hardware features representing dynamic profiling of the benign and malware program. The steps involved in the victim HMD training are shown in Figure 1. In particular, we train a regularized logistic regression that uses 'liblinear' algorithm for optimizing cost function based on ℓ_2 penalization, maximum iteration of 2000, and tolerance of 0.0001. We use *victim training* dataset for training the baseline HMD and *testing* dataset to measure detection performance. Figure 2 shows the performance of our baseline HMD (victim HMD) using different performance metrics such as accuracy (% of total program samples correctly classified), sensitivity (% of malware samples correctly classified), specificity (% of regular program samples correctly classified), precision (% of predicted malware labels that are actually malware), and F1-score (harmonic mean of sensitivity and precision). We choose logistic regression for its simplicity, yet state-of-the-art performance, low cost and complexity of its hardware implementation. Note that the victim HMD will be used as our baseline HMD throughout the remaining experiments.

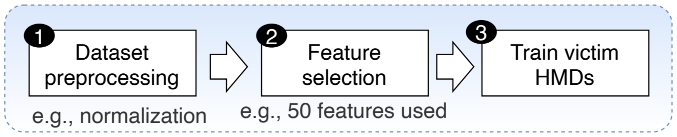

Fig. 1. Steps involved in training victim HMDs.

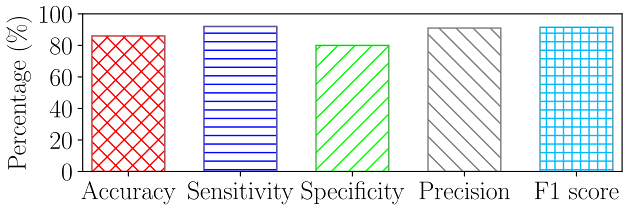

Fig. 2. Baseline HMD (vicitm HMD) detection performance.

B. The threat model

As stated earlier, we assume two threat models; black-box and white-box attacks. In this section, we explain how we generate these attacks and subsequently show that the current generation of HMDs are vulnerable to both of these attacks scenarios. Figure 3 shows the steps involved in generating attacks. In particular, in black-box attacks the attacker starts from step 1, while in the white-box attack scenario the attacker starts from step 2.

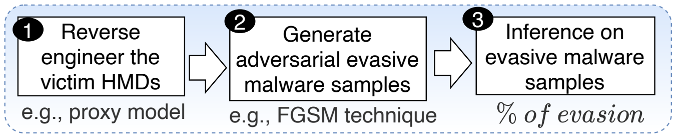
1 Reverse engineer the victim HMDs
e.g., proxy model
2 Generate adversarial evasive malware samples
e.g., FGSM technique
3 Inference on evasive malware samples
% of evasion

Fig. 3. Steps involved in generating attack.

(1) Black-box attack (reverse-engineering HMDs): In this threat model, we assume that attackers have black-box access to the HMD, which means that attackers can query the HMD and observe the detection/classification output. In addition, attackers do not know the HMD model's internal architecture or hyperparameters. However, generating evasive malware samples require HMD model information, otherwise the problem becomes NP-Hard [20]. Therefore, attackers need to reverse-engineer the victim HMD and build a substitute (proxy) model that can be used to generate evasive malware samples that can evade the victim HMD detection. For this purpose, attackers first prepare a dataset consisting of malware and benign programs; then the dataset is used to query the victim HMD, which produces corresponding output labels. Next, the dataset and the labels predicted by victim HMD are used to train a substitute model that we call the reverse-engineered HMD. At this stage, it is important to measure the reverse-engineering effectiveness, which implies how successful the reverse-engineering is, i.e., similarity between reverse-engineered HMD and the victim HMD detection behaviour. To measure the reverse-engineering effectiveness, attackers build another dataset (that we call a validation set, which should be different from the training dataset), which is used to query both the victim HMD and reverse-engineered HMD. Finally, the percentage of matched detection results, i.e., output labels, for both the reverse-engineered HMD and the victim HMD represents the reverse engineering effectiveness.

To show that current HMDs can be reverse-engineered, we used the *attacker training* dataset and query the victim HMD (described in Section III-A). Then the dataset and predicted labels were used to train a new logistic regression classifier, which is our reverse engineered model. We used *testing* dataset to evaluate the reverse engineering effectiveness. Our experimental result shows a reverse engineering effectiveness of 97%, which indicates that it is possible to reverse engineer the current generation of HMDs. After reverse engineering, the attacker use the reverse-engineered HMD to generate evasive malware programs, following steps 2 and 3 of Figure 3; these two steps are detailed in the white-box attack description. However, for black-box attacks, the reverse-engineered HMD is used in the second step instead of victim HMD.

The black-box attack success rate is measured through the transferability metric, which is the percentage of evasive malware that is generated to evade the reverse-engineered HMD that can also evade the victim HMD. To measure the black-box attacks transferability, we used the *testing* dataset to create the evasive malware samples. Figure 4 shows the transferability results while increasing the attack intensity, i.e., amount of added instructions. The results shows that the success rate of the black-box attack, i.e., percentage of malware that can evade the victim HMD's detection, increases while increasing the attack intensity.

(2) White-box attack (generating evasive malware): While the black-box attack scenario assumes that attacker does not know the victim HMD architecture and hyperparameters, the white-box attack scenario assumes that the attacker knows the victim HMD architecture and hyperparameters. Thus, it is more difficult to defend against white-box attack scenarios [7].

To develop evasive malware samples that can by-pass

HMDs detection, attacker have to add perturbations to the malware features, which requires identifying the features that need to be perturbed and embedding those features into malware program.

In our experiments, we identify the target features by using a slightly modified Fast Gradient Sign Method (FGSM) algorithm [3], which is primarily introduced for image classification domain. Original FGSM allows bi-directional feature perturbation, i.e., image features can be increased or decreased. However, we slightly modify FGSM by imposing constraints that allow unidirectional feature perturbation, i.e., features can be only increased. We need such modification since HMDs' features follow a positive continuous distribution, i.e., cannot have a negative value. As a results the only way to develop evasive malware samples is by inserting features into malware program. Assuming x to be the input to HMD, y be the corresponding output, and $L(\theta, x, y)$ be the loss function; then the features that need to be perturbed are identified by the gradient of the loss function. Note that our features are executed instruction mixes, as described in Section II. Once the features (instructions) are identified, we embed them in malware programs by following an approach similar to [7]. We cannot directly insert instructions into malware programs since our malware samples are only binary executable; also, we do not have access to their source code nor can we decompile them as they are obfuscated. Therefore, we followed an alternative approach by (i) constructing the dynamic control flow graph (DCFG) of each malware using Intel Pin tool, and (ii) inserting required instructions into each basic block of the DCFG. While inserting instructions, we make sure that they do not alter the state of the program, i.e., malware can perform their intended malicious activities.

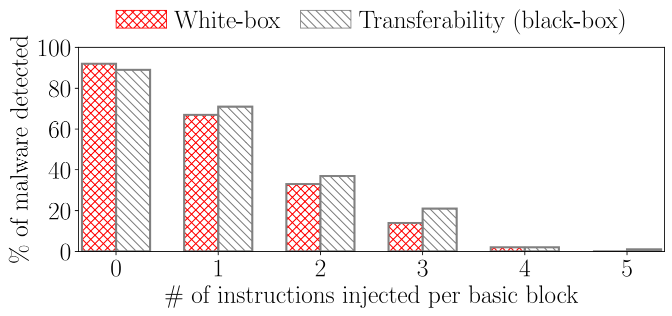

Fig. 4. Adversarial evasion attacks effectiveness on current HMDs

It is expected that inserting instructions would turn the malware into evasive, which would evade victim HMD from detection, resulting in a drop in the HMD detection accuracy. At this stage, we evaluate the effectiveness of white-box and black-box attacks by developing evasive samples based on the *testing* dataset, and the results are shown in Figure 4. As we can see, inserting one instruction per basic-block drops the victim HMD malware detection from 92% to 67% in case of white-box attack, and proxy HMD malware detection drops from 89% to 71% in case of black-box attack. Furthermore, inserting more and more instructions per basic-block decreases the malware detection rates. Therefore, the obtained results confirm that the current generation of HMDs are vulnerable to white-box attack.

IV. Monotonic-HMDs

In this section, we first define monotonic classifier. Subsequently, we discuss the rationale behind using monotonic model to harden HMDs against adversarial evasion attacks. Finally, we describe how we constructed our Monotonic-HMDs.

Monotonic classifier: Monotonic classifier are classifiers that imposes monotonicity constraints on a model; a function $f : x \rightarrow y$ is called increasingly monotonic if $x \leq x^{'} \implies f(x) \leq f(x^{'})$. Likewise, consider f as an $m-ary$ classifier that operates on $d-dimensional$ feature vector $x_1, x_2, x_3, \ldots, x_d$, i.e., $f : x \in R^d \rightarrow y \in N^m$; also suppose that x_i and $x_i^{'}$ are two different values of $i-th$ feature, where $i \in \{1, 2, 3, \ldots, d\}$. Then classifier f is increasingly monotonic if $x_i \leq x_i^{'}$ implies $f(x_1, x_2, \ldots, x_i, x_{i+1}, \ldots, x_d) \leq f(x_1, x_2, \ldots, x_i^{'}, x_{i+1}, \ldots, x_d)$. In other words, monotonicity constraint for classifier ensures that increasing a feature value will only increase the model's output value.

Monotonic-HMDs against adversarial evasion attacks: To develop evasive malware samples that can evade HMDs' detection, attackers usually target benign features that have positive correlation with the output of the HMD being a benign. Thus, by simple increasing their value, i.e., adding more of them into the malware, the malware can evade detection. The rational behind targeting benign features is that it is easy to manipulate without changing the malicious functionality of the malware. As a result, all previously proposed adversarial attacks target increasing the benign features values to craft evasive malware samples [2], [4], [7], [8].

Furthermore, attackers usually seek to have their attacks be misclassified as benign; conversely, there is less incentive to make benign applications be misclassified as attacks. Therefore, we explore taking advantage of this fundamental asymmetry in both the features and the attacker goals, to construct a monotonic classifier. Specifically, monotonic classifier can enforce monotonic constraints such that increasing the value of a feature would result in a higher probability of getting detected. Therefore, to build such a classifier, we first need to identify the benign features that can be an easy target for the attackers to create evasive malware. After identifying the benign features, we train monotonic constrained models using the selected features only. Consequently, if the attacker tries to change any feature, increasing the feature value will only result in higher chance of being detected.

Monotonic-HMDs construction: Figure 5 explains how we constructed Monotonic-HMDs. At first, we train a logistic regression classifier using the dataset described in Section II. Then we identify the malicious and benign features; malicious features contribute to the HMDs decision making toward detecting a running process as malware, and benign features contribute to the probability of a process being benign.

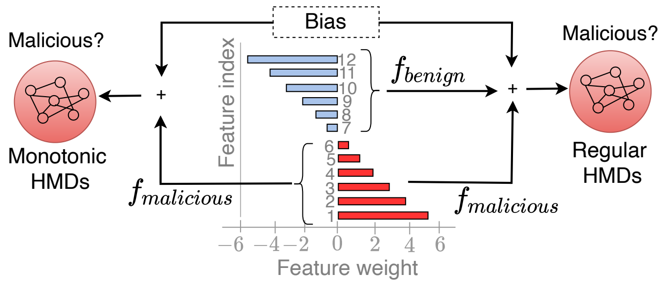

Fig. 5. Monotonic-HMDs

As such, features having positive weights are identified as malicious, and features with negative weights are identified as benign; we denote malicious features by $f_{malicious}$ and benign features as f_{benign} in Figure 5. As shown in Figure 5, Monotonic-HMDs discard benign features but comprise only the malicious features and bias. Specific to our evaluation, the total number of features in our dataset is 50. After training a logistic regression classifier using the 50 features, the classifier contains 25 positive weights, i.e, malicious features, and 25 negative weights, i.e., benign features. Since we construct the Monotonic-HMD using the malicious features only, the resulting classifier, i.e., Monotonic-HMD, will use less number of features compared to the regular HMDs. Therefore, we will study the Monotonic-HMD detection performance, speed, and model size (Section V) as well as their hardware implementation overhead (Section VI).

V. Experimental Evaluation

In this section, we first perform security evaluation of the proposed Monotonic-HMDs. Subsequently, we evaluate the Monotonic-HMD's detection performance and implementation overhead. Specifically, we study their detection accuracy, sensitivity, inference speed, and model size. *Accuracy* shows how many programs the HMD correctly labeled out of all programs; *sensitivity* shows how many malware the HMD correctly classified out of all programs that were labeled as malware, e.g., sensitivity = $x\%$ indicates that $x\%$ evasive malware samples were detected and $100 - x\%$ malware samples evaded the HMDs from detection.

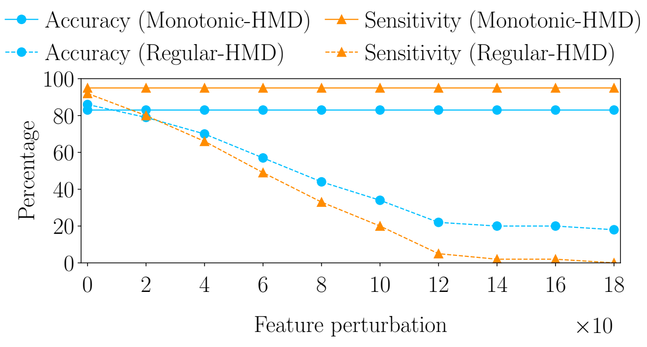

Fig. 6. Performance of Monotonic-HMD and regular HMD

Figure 6 shows the detection accuracy and sensitivity for various attack intensity. As stated earlier, evasive malware samples are generated by perturbing features; thus, amount of feature perturbation indicates attack intensity— feature perturbation = 0 means their is not perturbations to any features and thus, the malware is not evasive and any higher perturbation indicates more aggressive attack. Looking at no-attack scenario, i.e., non-evasive malware, regular HMD shows 86% accuracy and 92% sensitivity, whereas monotonic HMD shows 83% accuracy and 95% sensitivity, i.e., monotonic HMD loses 3% accuracy but gains 3% on sensitivity; this is because monotonic HMDs discard benign features, as shown in Figure 5.

As the attack intensity increases, regular HMD accuracy and sensitivity continuously drops, which is caused by the positive correlation between the benign features, i.e., perturbed features, and the HMD's output being benign. Thus, larger proportion of evasive malware samples are able to evade regular HMD as we increase the attack intensity. And 0% sensitivity at feature perturbation of 180 indicates that the regular HMDs are completely fooled, where all the malware samples evaded from detection.

On the other hand, the monotonic-HMD maintain the stable accuracy and sensitivity despite increasing attack intensity, which means feature perturbation cannot confuse or evade monotonic HMDs. This is due to the attacks strategies proposed in previous works [2], [4], [7], [8].

TABLE I
Speed and size comparison of Monotonic-HMDs and regular HMDs

	Inference time	**Model size**
Regular HMDs	0.36 (μs)	1024 (Bytes)
Monotonic-HMDs	0.14 (μs)	901 (Bytes)
Savings by Monotonic-HMDs	61.11%	12.01%

In addition, we evaluate the monotonic-HMDs detection speed and model size. Table I shows that regular HMDs take 0.36 (μs) for detecting one sample, whereas monotonic-HMDs take only 0.14 (μs), indicating 61.11% reduction in inference time. Moreover, monotonic-HMDs reduce the memory space needed to store the model since the model size of monotonic-HMDs is 12.01% less compared to regular HMDs. Overall, monotonic HMDs ensures faster malware detection with reduce model size since it has less features, i.e., only use the malicious features.

VI. Hardware Implementation Evaluation

We extended the open-source x86 processor (AO486), which is a 32-bit in-order pipelined implementation of the Intel 80486 ISA, once with regular-HMD and once with the proposed Monotonic-HMDs, to evaluate the hardware implementation overhead of the Monotonic-HMD implementation. Both HMDs (Monotonic-HMDs and regular-HMDs) were added to the end of the CPU pipeline, e.g., after the commit stage, and were synthesized on Xilinx Virtex-7 VC707 Evaluation Board by Vivado 2017.4. The hardware implementation overhead results are shown in Table II. The results demonstrated that Monotonic-HMDs offers both area and power savings, around 50% in our case study, compared to regular HMDs.

TABLE II
HMDS' EFFECT ON CORE

	Regular-HMDs	**Monotonic-HMDs**
Logic Cells	+0.28%	+0.136%
Power Usage	+0.08%	+0.042%

VII. Discussion

We investigate the use of the emerging monotonic models for HMDs as a defense against adversarial evasion attacks. The addition of monotonic constraint allows the HMDs to leverage malicious features rather than benign features in their detection model. Thus, previously proposed attacks [2], [4], [7], which depend on targeting the benign features by increasing their presence in the malware program to evade detection, can not evade monotonically constraint HMDs. Nonetheless, as a cost for security, an overall degradation of 3% in detection accuracy is observed when using monotonically-constrained HMDs, i.e., Monotonic-HMDs, compared to regular HMDs under no evasions attempt.

Furthermore, while Monotonic-HMDs are robust against previously proposed attacks [2], [4], [7], [8], we believe future attacks that target the malicious features only are possible and can evade Monotonic-HMDs. However, such future attacks would require advanced domain expertise and non-trivial efforts by the attacker because the evasion strategy should be malware specific so that the malicious functionality is not effected. Nonetheless, we hope that Monotonic-HMDs raises the bar for adversary to evade detection since simple evasion strategies are no longer possible.

VIII. Concluding Remarks

In this work, we proposed Monotonic-HMDs as a defence against adversarial evasive malware attack. We construct Monotonic-HMDs by imposing monotonicity constraints on regular HMDs model and features. We showed that at attack intensity where current HMDs are completely fooled and evaded, Monotonic-HMDs achieve and maintain high robustness accuracy, turning it immune to evasion attack. Moreover, Monotonic-HMDs offer $2.57\times$ faster malware detection and reduce model size by 12.01% compared to current generation of HMDs. Furthermore, our hardware implementation shows that Monotonic-HMDs offer 50% savings in area and power usage, over regular HMDs, due to including reduced number of features in the model.

References

[1] Demme, J., Maycock, M., Schmitz, J., Tang, A., Waksman, A., Sethumadhavan, S., and Stolfo, S. On the feasibility of online malware detection with performance counters. *ACM SIGARCH Computer Architecture News 41*, 3 (2013), 559–570.

[2] Dinakarrao, S. M. P., Amberkar, S., Bhat, S., Dhavlle, A., Sayadi, H., Sasan, A., Homayoun, H., and Rafatirad, S. Adversarial attack on microarchitectural events based malware detectors. In *DAC* (2019), ACM, p. 164.

[3] Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[4] Islam, M. S., Kuruvila, A. P., Basu, K., and Khasawneh, K. N. Nd-hmds: Non-differentiable hardware malware detectors against evasive transient execution attacks. In *ICCD* (2020), IEEE, pp. 537–544.

[5] Kayaalp, M., Khasawneh, K. N., Esfeden, H. A., Elwell, J., Abu-Ghazaleh, N., Ponomarev, D., and Jaleel, A. Ric: Relaxed inclusion caches for mitigating llc side-channel attacks. In *DAC* (2017), IEEE, pp. 1–6.

[6] Kazdagli, M., Huang, L., Reddi, V., and Tiwari, M. EMMA: A new platform to evaluate hardware-based mobile malware analyses. *CoRR abs/1603.03086* (2016).

[7] Khasawneh, K. N., Abu-Ghazaleh, N., Ponomarev, D., and Yu, L. Rhmd: evasion-resilient hardware malware detectors. In *MICRO* (2017), IEEE, pp. 315–327.

[8] Khasawneh, K. N., Abu-Ghazaleh, N. B., Ponomarev, D., and Yu, L. Adversarial evasion-resilient hardware malware detectors. In *ICCAD* (2018), IEEE, pp. 1–6.

[9] Khasawneh, K. N., Ozsoy, M., Donovick, C., Abu-Ghazaleh, N., and Ponomarev, D. Ensemble learning for low-level hardware-supported malware detection. In *RAID* (New York, NY, USA, 2015), Springer-Verlag New York, Inc., pp. 3–25.

[10] Khasawneh, K. N., Ozsoy, M., Donovick, C., Ghazaleh, N. A., and Ponomarev, D. V. Ensemblehmd: Accurate hardware malware detectors with specialized ensemble classifiers. *IEEE Transactions on Dependable and Secure Computing* (2018).

[11] Luk, C.-K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V. J., and Hazelwood, K. Pin: building customized program analysis tools with dynamic instrumentation. In *Acm sigplan notices* (2005), vol. 40, ACM, pp. 190–200.

[12] Moser, A., Kruegel, C., and Kirda, E. Limits of static analysis for malware detection. In *ACSAC* (2007), IEEE, pp. 421–430.

[13] Ozsoy, M., Donovick, C., Gorelik, I., Abu-Ghazaleh, N., and Ponomarev, D. Malware-aware processors: A framework for efficient online malware detection. In *HPCA* (2015), IEEE, pp. 651–661.

[14] Ozsoy, M., Khasawneh, K. N., Donovick, C., Gorelik, I., Abu-Ghazaleh, N., and Ponomarev, D. V. Hardware-based malware detection using low level architectural features.

[15] Patel, N., Sasan, A., and Homayoun, H. Analyzing hardware based malware detectors. In *DAC* (2017), ACM, p. 25.

[16] Qualcomm smart protect technology, 2016. Last Accessed July 2016 from https://www.qualcomm.com/products/snapdragon/security/smart-protect.

[17] Sayadi, H., Houmansadr, A., Rafatirad, S., Homayoun, H., et al. Comprehensive assessment of run-time hardware-supported malware detection using general and ensemble learning. In *ICCF* (2018), ACM, pp. 212–215.

[18] Sayadi, H., Patel, N., PD, S. M., Sasan, A., Rafatirad, S., and Homayoun, H. Ensemble learning for effective run-time hardware-based malware detection: A comprehensive analysis and classification. In *DAC* (2018), IEEE, pp. 1–6.

[19] Tang, A., Sethumadhavan, S., and Stolfo, S. J. Unsupervised anomaly-based malware detection using hardware features. In *RAID* (2014), pp. 109–129.

[20] Vorobeychik, Y., and Li, B. Optimal randomized classification in adversarial settings. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems* (2014), International Foundation for Autonomous Agents and Multiagent Systems.

[21] Younis, Y. A., Kifayat, K., Shi, Q., and Askwith, B. A new prime and probe cache side-channel attack for cloud computing. In *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing* (2015), IEEE, pp. 1718–1724.

[22] Yuval Nativ, Lahad Ludar, f. theZoo - the most awesome free malware database on the air, 2015. Available online (last accessed, November 2019): https://thezoo.morirt.com.