Recent Advancements in Microarchitectural Security: Review of Machine Learning Countermeasures

Hossein Sayadi*, Han Wang‡, Tahereh Miari*, Hosein Mohammadi Makrani‡,
Mehrdad Aliasgari*, Setareh Rafatirad†, Houman Homayoun‡

*California State University, Long Beach, †George Mason University, ‡University of California, Davis
*{hossein.sayadi,tahereh.miari01,mehrdad.aliasgari}@csulb.edu, †{srafatir}@gmu.edu, ‡{hjlwang,hmakrani,hhomayoun}@ucdavis.edu

***Abstract*—Recent microarchitectural security countermeasures by employing applications' low-level features collected from Hardware Performance Counters (HPCs) registers have emerged as a promising solution to address the inefficiency of traditional software-based methods. Furthermore, recent studies have shown that malicious activities at the hardware level ranging from application-based malware to microarchitecture-based Side-Channel Attacks (SCAs) can be accurately distinguished from normal traces using Machine Learning (ML) algorithms. Such ML-based countermeasures reduce the latency of attack detection process as well as the hardware and resource utilization overheads. This paper provides an in-depth analysis of recent advancements in machine learning countermeasures for microarchitectural security to detect malicious software and emerging side-channel attacks.**

Keywords: Microarchitectural Security, Machine Learning; Malware; Side-Channel Attacks; Hardware Performance Counters

I. Introduction

Cybersecurity for the past decades has been in the front line of global attention as a critical threat to the information technology infrastructures. Attackers are increasingly motivated and enabled to compromise software and computing hardware infrastructure. Recent studies have shown that the attackers take advantage of emerging hardware vulnerabilities to compromise systems and deploy malicious activities. Hence, the security of a computer system can be compromised at the hardware level through various types of attacks such as by executing malicious applications to infect the target host or deploying microarchitectural side-channel attacks (SCAs) to infer confidential information. Malware refers to any piece of software written with the intent of stealing data, unauthorized data access, damaging devices, etc. Viruses, Trojans, Spyware, Rootkits, and Ransomware are among the different types of malware [1, 2]. Microarchitectural SCAs have also posed serious threats to the security of modern computing systems. Such attacks exploit side-channel vulnerabilities originating from fundamental performance improving components such as cache memories [3, 4].

The significant growth of modern computing systems in embedded applications and Internet-of-Things (IoT) domains has further highlighted the serious impact of such threats. There exists some important factors influencing the security vulnerability of embedded systems and IoTs including the limited energy and resources available, the low computational capacity, and significant number of computing nodes in the network [5, 6]. Therefore, to keep on combating the increase in malicious cyber-attacks, there is an urgent need to develop intelligent security countermeasures to protect the integrity and confidentiality of the authenticated users' information.

Conventional software-based detection techniques have shown to be inefficient mostly imposing significant complexity and computational overheads on the system. In addition, such detection methods depend on the static signature analysis of executed applications that makes them incapable of detecting complex unknown attacks. Recent advancements in microarchitectural security have demonstrated that malicious activities at the processor hardware level ranging from application-based malware to microarchitecture-level side-channel attacks can be accurately identified with the aid of standard Machine Learning (ML) algorithms [2, 7, 8]. Such methods apply the standard ML techniques on the low-level features such as microarchitectural events collected by Hardware Performance Counter (HPCs) registers to train and test effective classifiers to detect malicious patterns of running programs. HPC events have primarily been deployed to conduct architectural performance analysis and tuning. Recent works have proposed to utilize the HPCs information for securing the hardware systems against both malware software and microarchitectural SCAs.

Microarchitectural security in the form of detection of malicious software and emerging side-channel attacks in computing systems, is an area of major concern not only to the research community but also to the general public. This paper devotes to the comprehensive analysis of machine learning-focused defense mechanisms against malicious applications and SCAs targeting the available on-chip hardware performance counters in modern microprocessors. In particular, we present a review of recent ML-based detection mechanisms and countermeasures for enhancing the security of the systems at the microarchitecture level.

II. Deployment of Hardware Performance Counters for Security

The complexity of today's computing systems has tremendously increased in the past decades. Hierarchical cache subsystems and processor pipeline, simultaneous multithreading, and out-of-order execution units have a significant impact on the performance of computing systems. Access to the performance monitoring module, an essential feature in modern microprocessors (e.g. Intel, ARM, and AMD), is generally provided in the form of programmable hardware performance counter registers. HPCs are specialized registers designed inside modern microprocessors to monitor and capture different hardware-related events [9, 10]. Due to limited number of physical expensive to implement HPC registers on the processor chip, HPCs are constrained in the number of events that could be counted concurrently.

HPCs are able to count a variety of low-level events such as cache memories access and misses, TLB hits and misses, and branch mispredictions for various optimization targets such as performance, energy-efficiency, and security enhancement.

TABLE I: Common HPC features used for malware detection

HPC event	Description
Branch instructions	# branch instructions retired
Branch-misses	# branches mispredicted
Instructions	# instructions retired
bus-cycles	time to make a read/write between the cpu and memory
Cache misses	# last level cache misses
Cache-references	# last level cache references
L1-dcache-load-misses	# cache lines brought into L1 data cache
L1-dcache-loads	# retired memory load operations
L1-dcache-stores	# cache lines into L1 cache from DRAM
L1-icache-load-misses	# instruction misses in L1 instructions cache
node-loads	# successful load operations to DRAM
node-stores	# successful store operations to DRAM
LLC-load-misses	# cache lines brought into L3 cache from DRAM
LLC-loads	# successful memory load operations in L3
iTLB-load-misses	# misses in instruction TLB during load operations
Branch-loads	# successful branches

Table I reports some of the commonly deployed low-level features captured by HPC registers from Perf tool under Linux in a recent hardware-based malware detection work [2].

III. Application of Machine Learning for Security Countermeasures

Recent developments in the area of machine learning and data mining, motivated by a significant increase in the size of data from high-performance computing systems, have resulted to successful applications of ML in various domains such as security enhancement of computing systems. Figure 1 demonstrates a general overview of machine learning-based countermeasures for microarchitectural security in the form of malware or SCAs detection. As shown, it often includes different stages such as monitoring the application to profile the HPC data, feature analysis, training, and testing the ML-based detector using the collected features. The ML models trained by low-level microarchitectural features continuously learn by analyzing the HPCs data to identify the malicious patterns and protect the processor architecture against information leakage caused by emerging SCAs.

A. *Feature Selection: Key Microarchitectural Features*

Identifying the prominent low-level features is an important step for developing accurate ML-based countermeasures. There exists numerous microarchitectural events with different functionality available to collect from running programs in modern microprocessors. Counting all possible features would result in a high dimensional data which increases computational complexity and induces delay. Moreover, including irrelevant features could reduce the accuracy of classifiers [11, 12].

As a result, feature selection methods are beneficial for enhancing the performance of learning process, reducing the computational complexity and the required storage on the computing systems. For effective run-time detection in resource-limited systems (e.g. embedded devices) which have limited number of HPCs physically available on the processor's chip, feature selection even plays a more important role in determining the minimal set of critical HPCs to collect the required data in a single run [2]. The selected HPC features are then used to train each ML-based detector in which the classifier attempts to find a correlation between the feature values and the application behavior to predict the existence of malicious patterns (benign or attack type).

B. *Performance Evaluation Metrics*

Evaluating the performance of machine learning classifiers is an important step in implementing effective ML-based countermeasure techniques. In machine learning and statistics,

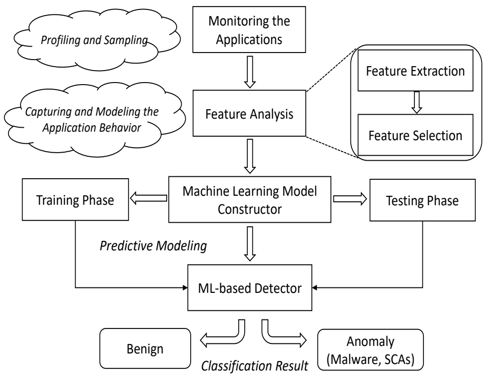

Fig. 1: General overview of machine learning countermeasures for malware and SCAs detection

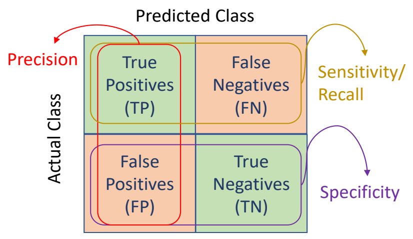

Fig. 2: Confusion matrix layout for a machine learning classifier

there are variety of measures that can be deployed to evaluate the performance of a detection method in order to show its detection accuracy. Confusion matrix is a basic evaluation measure for this purpose. In statistical machine learning domain, a confusion matrix, is a specific table with four outcomes produced as a result of binary classification that represents the prediction performance of a classifier. As depicted in Figure 2, a confusion matrix comprised of two dimensions namely as "actual" and "predicted", and identical sets of "classes" in both dimensions. Each row of the confusion matrix represents the instances in a predicted class while each column represents the instances in an actual class.

The standard evaluation metrics used for performance analysis of ML-based security countermeasures are summarized in Table II. For analyzing the detection rate, malicious applications samples are considered as positive instances. Hence, the True Positive Rate (TPR) represents the proportion of correctly identified positive instances or malicious samples. The True Negative Rate (TNR) also evaluate the specificity that measures the proportion of correctly identified bengin files or negative samples. In addition, the False Positive Rate (FPR) is the rate of benign files that are wrongly classified as malware.

The F measure (F score) in ML is interpreted as a weighted average of the precision (p) and recall (r). The precision is the proportion of the sum of true positives versus the sum of positive instances and the recall is the proportion of instances that are predicted positive of all the instances that are positive. F measure is a more comprehensive evaluation metric over accuracy (percentage of correctly classified samples) since it takes both the precision and the recall into consideration. More importantly, F measure is also resilient to class imbalance in the dataset which is the case in our experiments. The Detection Accuracy (ACC) further measures the rate of the correctly classified positive and negative samples [9, 13].

Receiver Operating Characteristic (ROC) is a statistical

TABLE II: Evaluation metrics for performance of ML-based detection

Evaluation Metric	Description
True Positive (TP)	Correct positive prediction
False Positive (FP)	Incorrect positive prediction
True Negative (TN)	Correct negative prediction
False Negative (FN)	Incorrect negative prediction
Specificity: True Negative Rate	$TNR = TN/(TN + FP)$
False Positive Rate	$FPR = FP/(FP + TN)$
Precision	$P = TP/(FP + TN)$
Recall: True Positive Rate	$TPR = TP/(TP + FN)$
F measure (F score)	$Fmeasure = 2 \times (P \times R)/(P + R)$
Detection Accuracy	$ACC = (TP + TN)/(TP + FP + TN + FN)$
Error Rate	$ERR = (FP + FN)/(P + N)$
Area Under the Curve	$AUC = \int_0^1 TPR(x)dx = \int_0^1 P(A > \tau(x))dx$

plot that depicts a binary detection performance while its discrimination threshold setting is changeable. The ROC space is supposed by FPR and TPR as x and y axes, respectively. It determines trade-offs between TP and FP (the benefits and costs analysis). Given that the TPR and FPR are equivalent to sensitivity and (1-specificity) respectively, each prediction result represents one point in the ROC graph in which the point in the upper left corner ([0, 1]) stands for the perfect detection result, indicating 100% sensitivity and 100% specificity. Area Under the Curve (AUC) is another important evaluation metric for checking any ML model's performance at various thresholds settings. It shows how well a classification model is capable of distinguishing between different classes.

IV. Microarchitectural Security

This section presents the state-of-the-arts on malware and side-channel attack detection using machine learning techniques at processor microarchitecture level. Table III further characterizes recent hardware-assisted malware and SCA detection techniques and their deployed classification methods.

A. Malware Detection using HPCs information

In response to the challenges of traditional malware detection solutions, Hardware-assisted Malware Detection (HMD) methods were proposed that use low-level features captured by HPCs combined with ML techniques to distinguish the malicious patterns from benign applications. The recent studies have also demonstrated the efficacy of HMD methods in lowering the computational latency and hardware overheads of detection process [2]. The work in [8] was the first proposal that explored the feasibility of deploying HPCs information for building accurate ML-based malware detectors. They applied ML algorithms such as Neural Network and K-Nearest Neighbour and demonstrated high detection accuracy for Android malware detection. Tang et al. [14] further discussed the application of unsupervised learning on low-level hardware-related data to identify buffer overflow attacks. In a different work, Ozsoy et al. [15] deployed sub-semantic features to implement Logistic Regression (LR) and Neural Network-based classifiers for real-time malware detection. While effective, their proposed solution suggested modification in microprocessor pipeline which increases the overhead and complexity.

The research in [16] applied logistic regression to classify malware into different types and trained a specialized classifier for detecting each type. The work in [10] proposed accurate ML model to detect synthetic rootkits based on HPCs features. To address the challenges of HPC-based run-time malware detection, Sayadi et al. [2] proposed ensemble learning techniques to improve the performance of hardware-assisted malware detectors by accounting for the impact of reducing the number of HPC features on the performance of malware detectors. In addition, a recent work in [9] proposed a two-stage machine learning-based approach for run-time malware detection in which in the first level classifies applications using a multiclass classification technique into either benign or one of the different malware classes In the second level, to have a high detection performance, the authors deploy an ML model that works best for each class of malware and further apply effective ensemble learning to enhance the performance HMD.

B. Side-Channel Attack Detection using HPCs information

With increasing computation demand in modern computer systems, different components such as cache memories, branch predictors, and out-of-order execution units are designed in processors to enhance the performance. Nevertheless, recent studies have shown that such solutions have resulted in new microarchitectural vulnerabilities which could be exploited in modern processors. Cache-based SCAs have demonstrated powerful capabilities of stealing users' critical information (e.g secret keys of cryptographic applications) within the same processing core or cross-core residency of victim applications.

Flush+Reload: Flush-Reload [20, 21] exploits the weakness of page de-duplication and monitors memory access lines in shared pages. This attack flushes out the victim data in the cache and waits for the victim execution. The attacker then reloads data by accessing them and measures the accessing time. If accessing time is shorter, it infers the data is accessed by the victim; else, it has not been accessed by the victim.

Flush+Flush: This attack relies on the execution time of the flush instruction. Unlike prior attacks, Flush-Flush does not make any memory accesses, nor it relies on the access latency of the data. In particular, in this type of SCA, the setup and first stage is the same as Flush-Reload. In the second stage, instead of reloading the shared memory blocks, the adversary flushes the blocks. If the victim fetches a block into the cache, then flushing this block will take a longer time than when it is out of the cache. Hence, the Flush has the same effect as Reload. Moreover, a single Flush operation can serve as a Check for the current round as well as Set for the next round.

Prime+Probe: This attack targets at L1/L3 data caches. Prime Probe attack consists of two stages including Prime and Probe. In the Prime stage, the attacker builds an eviction set (a group cache sets causing potential conflict with victims) and fills cache with the eviction sets. Next, the attacker waits for victim execution and then re-accesses the eviction sets (Probe stage). If the accessing time is long enough, it means the victim accessed the data; else, the victim does not access the data [22].

Spectre Attack: Recent Spectre attack takes exploit speculative execution by locating the instructions firstly and tricking the CPU into speculatively and erroneously executing this instruction sequence, which leaks information [23]. Speculative execution is used in commercial processors to boost performance by executing the next execution path predicted by control flow [24]. When CPU waits for data coming from memory or disk, the current register state is stored and then the speculative instruction is executed.

The work in [25] proposed an HPC monitoring model to detect the SCAs collected from both victims and attacks applications. Similarly, in [17] the authors presented CloudRadar which aims at detecting cross-VM side-channel attacks by deploying HPC patterns. The research in [26] offered a detection system containing one analytic server and one or more monitored computing devices to detect SCAs. The

TABLE III: Summary of recent hardware-assisted malware and side-channel attack detection techniques and their classification methods

Research	Platform	Classification Model	Threat Type	Microarchitectural Features	Evaluation Metric
[8]	Android, Linux	KNN, NN, DT, RF	Malware	Low-level hardware performance counters in the form of multi-dimensional time series data	FP, ROC, AUC
[14]	Linux	ocSVM	Malware	22 features including LLC, Load & store instructions retired, Branch instructions retired, etc.	F Score, AUC, ROC
[15]	Windows	LR, NN	Malware	Instruction mix features, Memory reference patterns, and Architectural events	ACC, S, C, FP, ROC
[16]	Windows	LR, NN, EL	Backdoor, PWS, Rogue, Trojan, Worm	Instruction mix features, Memory reference patterns, and Architectural events same as [15]	ACC, FP, ROC, AUC
[10]	Linux	SVM, ocSVM, NB, DT	Kernel Rootkits	8 low level events (branch instructions, cache misses, etc.)	Confusion Matrix, ROC
[2]	Linux	BN, J48, JRip, MLP, OneR, RT, SGD, SMO, AB, BG	Malware	(32/16/8/4/2) low-level events (branch instructions, cache misses, etc.)	ACC, AUC, ACC*AUC, HWO
[9]	Linux	J48, JRip, MLP, OneR, AB	Virus, Trojan, Rootkits, Backdoor	8/4 low-level events (branch instructions, cache misses, etc.)	F Score, AUC, F Score*AUC, HWO
[17]	Linux	DTW	Flush+Reload, Prime+Probe	16 low-level features (instructions, branch instructions, mispredicted branch instructions, etc.	ACC, F Score, ROC
[18]	Linux	LDA, LR, and SVM	Flush+Reload, Flush+Flush	4 low-level features (L1 data cache misses, L1 instruction cache accesses, L3 cache misses and cycles	ACC, FP, ROC
[3]	Linux	CPD	Flush+Reload, Flush+Flush, Prime+Probe	7 low-level features (PAPI_L3_TCM, Cycles, PAPI_REF_CYC, PAPI_CA_SNP, etc.)	ACC, FP
[19]	Linux	NN	Spectre Attacks	3 low-level features (L3 cache misses (L3_TCM), L3 cache accesses (L3_TCA) and total number of instructions (TOT_INS)	ACC, F Score, FP

Accuracy: ACC, Hardware Overhead: HWO, Sensitivity: S, Specificity: C, K Nearest Neighbor: KNN, BayesNet: BN, NaiveBayes: NB, Logistic Regression: LR, AdaBoost: AB, Bagging: BG, Support Vector Machine: SVM, One Class SVM: ocSVM, Neural Netework: NN, Last Level Cache References: LLC, REPTree: RT, Decision Tree: DT, Random Forest: RF, Ensemble Learning: EL, Dynamic Time Warping: DTW, Linear Discriminant Analysis: LDA, Change Point Detection theory: CPD.

analytic server receives HPCs data from monitored devices and identifies suspicious core activity. Then, an application level monitor is deployed to perform corrective actions. Other work in [27] proposed an online detection of Spectre by monitoring microarchitectural features using time series classification.

V. Conclusion

Hardware performance counter registers are special hardware units used for counting microarchitectural events in modern microprocessors. While cyber-attacks such as malicious software and Side-Channel Attacks (SCAs) are increasing in number and sophistication, the vast majority of them are targeting basic hardware signatures, such as missed branch predictions, cache misses, etc. to perform malicious activities. Recent advancements have demonstrated the application of Machine Learning (ML) techniques for detecting malware and emerging SCAs using HPCs profiles. In addition, emergence of adversarially crafted malicious software and complex SCAs highlights the importance of developing advanced analysis techniques for achieving a higher security. Efficient deployment of on-chip HPC registers is also imperative for accurate and cost-effective security enhancement. Therefore, this paper attempted to provide a comparative analysis of recent studies on microarchitectural security countermeasures for malware and SCAs detection using ML techniques.

References

[1] G. McGraw and G. Morrisett, "Attacking malicious code: A report to the infosec research council," *IEEE software*, pp. 33–41, 2000.

[2] H. Sayadi *et al.*, "Ensemble learning for effective run-time hardware-based malware detection: A comprehensive analysis and classification," in *Design Automation Conference (DAC)*. IEEE, 2018, pp. 1–6.

[3] S. Briongos, G. Irazoqui, P. Malagón, and T. Eisenbarth, "Cacheshield: Detecting cache attacks through self-observation," in *ACM Conference on Data and Application Security and Privacy*, 2018, pp. 224–235.

[4] H. Wang *et al.*, "Mitigating cache-based side-channel attacks through randomization: A comprehensive system and architecture level analysis," in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020, pp. 1414–1419.

[5] A. Mosenia and N. K. Jha, "A comprehensive study of security of internet-of-things," *IEEE TETC*, vol. 5, no. 4, pp. 586–602, Oct 2017.

[6] S. M. P. Dinakarrao *et al.*, "Lightweight node-level malware detection and network-level malware confinement in iot networks," in *DATE'19*, March 2019, pp. 776–781.

[7] H. Sayadi, "Machine learning-based solutions for secure and energy-efficient computer systems," Ph.D. dissertation, GMU, 2019.

[8] J. Demme *et al.*, "On the feasibility of online malware detection with performance counters," in *ISCA'13*, 2013, pp. 559–570.

[9] H. Sayadi *et al.*, "2smart: A two-stage machine learning-based approach for run-time specialized hardware-assisted malware detection," in *DATE'19*, March 2019, pp. 728–733.

[10] B. Singh *et al.*, "On the detection of kernel-level rootkits using hardware performance counters," in *ASIACCS'17*, 2017, pp. 483–493.

[11] H. Liu *et al.*, *Feature selection for knowledge discovery and data mining*. Springer Science & Business Media, 2012, vol. 454.

[12] H. M. Makrani *et al.*, "Energy-aware and machine learning-based resource provisioning of in-memory analytics on cloud," in *Proceedings of the ACM Symposium on Cloud Computing*, 2018, pp. 517–517.

[13] H. Sayadi *et al.*, "A data recomputation approach for reliability improvement of scratchpad memory in embedded systems," in *2014 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. IEEE, 2014, pp. 228–233.

[14] A. Tang *et al.*, "Unsupervised anomaly-based malware detection using hardware features," in *RAID'14*. Springer, 2014, pp. 109–129.

[15] M. Ozsoy *et al.*, "Malware-aware processors: A framework for efficient online malware detection," in *HPCA'15*, Feb 2015, pp. 651–661.

[16] K. N. Khasawneh and et al., "Ensemble learning for low-level hardware-supported malware detection," in *RAID'15*, 2015, pp. 3–25.

[17] T. Zhang *et al.*, "Cloudradar: A real-time side-channel attack detection system in clouds," in *RAID*. Springer, 2016, pp. 118–140.

[18] M. Mushtaq *et al.*, "Nights-watch: A cache-based side-channel intrusion detector using hardware performance counters," in *HASP*. ACM, 2018.

[19] J. Depoix and P. Altmeyer, "Detecting spectre attacks by identifying cache side-channel attacks using machine learning," *Advanced Microkernel Operating Systems*, p. 75, 2018.

[20] Y. Yarom and K. Falkner, "Flush+ reload: A high resolution, low noise, l3 cache side-channel attack." in *USENIX Security Symposium*, vol. 1, 2014, pp. 22–25.

[21] Y. Yarom, "Mastik: A micro-architectural side-channel toolkit," *Retrieved from School of Computer Science Adelaide: http://cs. adelaide. edu. au/~ yval/Mastik*, 2016.

[22] F. Liu *et al.*, "Last-level cache side-channel attacks are practical," in *SP*. IEEE, 2015, pp. 605–622.

[23] P. Kocher *et al.*, "Spectre attacks: Exploiting speculative execution," *arXiv preprint arXiv:1801.01203*, 2018.

[24] J. González and A. González, "Speculative execution via address prediction and data prefetching," in *Proceedings of the 11th international conference on Supercomputing*. ACM, 1997, pp. 196–203.

[25] M. Chiappetta *et al.*, "Real time detection of cache-based side-channel attacks using hardware performance counters," *Applied Soft Computing*, vol. 49, pp. 1162–1174, 2016.

[26] I. Prada *et al.*, "Detecting time-fragmented cache attacks against aes using performance monitoring counters," *arXiv:1904.11268*, 2019.

[27] C. Li and J. Gaudiot, "Online detection of spectre attacks using microarchitectural traces from performance counters," in *2018 SBAC-PAD*.