Comprehensive Evaluation of Machine Learning Countermeasures for Detecting Microarchitectural Side-Channel Attacks

Han Wang
University of California, Davis, CA, USA
hjlwang@ucdavis.edu

Hossein Sayadi
California State University, Long Beach, CA, USA
hossein.sayadi@csulb.edu

Avesta Sasan
George Mason University, Fairfax, VA, USA
asasan@gmu.edu

Setareh Rafatirad
George Mason University, Fairfax, VA, USA
srafatir@gmu.edu

Tinoosh Mohsenin
University of Maryland, Baltimore County, MD, USA
tinoosh@umbc.edu

Houman Homayoun
University of California, Davis, CA, USA
hhomayoun@ucdavis.edu

ABSTRACT

Microarchitectural Side-Channel Attacks (SCAs) have posed serious threats to the security of modern computing systems. Such attacks exploit side-channel vulnerabilities stemming from fundamental performance-enhancing components such as cache memories. The existing works on detection of SCAs based on low-level microarchitectural features have considered collecting both victim and attack applications' hardware events that are captured from processors' hardware performance counter (HPC) registers. However, in such techniques the attack HPCs data can be easily manipulated and/or corrupted resulting in misleading the SCAs detection mechanism. In addition, the prior studies have explored the suitability of a limited number of Machine Learning (ML) algorithms in detecting microarchitectural SCAs. In response, in this paper, we conduct a comprehensive evaluation of various machine learning-based countermeasures for real-time side-channel attack detection based on low-level microarchitectural features. For this purpose, the victim applications' behavior is collected using the HPC features and analyzed under no attack and attack conditions to avoid potential manipulation of attackers' HPCs. We further explore the HPCs monitoring overhead when microarchitectural features are sampled at different intervals to find out the appropriate sampling interval for SCAs detection. For the purpose of thorough analysis, various types of ML classifiers are implemented and precisely compared across different evaluation metrics including detection accuracy, F-measure, robustness (Area Under the ROC Curve), and computational latency to identify the most efficient ML classifiers for real-time microarchitectural SCAs detection.

KEYWORDS

Side-Channel Attack, Machine Learning, Microarchitectural Features, Real-time SCAs Detection

GLSVLSI '20, September 7–9, 2020, Virtual Event, China
© 2020 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-7944-1/20/09.
https://doi.org/10.1145/3386263.3407586

ACM Reference Format:
Han Wang, Hossein Sayadi, Avesta Sasan, Setareh Rafatirad, Tinoosh Mohsenin, and Houman Homayoun. 2020. Comprehensive Evaluation of Machine Learning Countermeasures for Detecting Microarchitectural Side-Channel Attacks. In *Proceedings of the Great Lakes Symposium on VLSI 2020 (GLSVLSI '20), September 7–9, 2020, Virtual Event, China.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3386263.3407586

1 INTRODUCTION

With increasing computation and performance demand in modern computer systems, various components (e.g. cache memories, branch predictors, out-of-order execution units, etc.) are implemented in processors architectures to minimize the CPU stalls and boost the performance. Despite the provided performance benefits, these solutions could cause new microarchitectural vulnerabilities which have been exploited by new type of attacks such as cache-based side-channel attacks that observe side-channel information by causing interference and infer sensitive information. Side-Channel Attacks (SCAs) primarily exploit hardware vulnerabilities to infer sensitive and confidential information [12, 19, 20]. Cache-based SCAs are one of the most common side-channel attacks that can be launched by the attacker remotely and require no physical access [9, 23]. The proliferation of computing devices in mobile and Internet-of-Things (IoT) domains further exacerbates the impact of emerging cybersecurity threats implying the necessity of protecting legitimate users from these attacks and calling for effective and low-cost security countermeasures. Hence, there exists an emerging need to address the security risks and challenges posed by such harmful attacks, calling for effective SCAs detection methodology which can accurately identify SCAs threats with minor overhead.

Recent studies on side-channel attack detection such as [4, 6, 24] examined the application of Machine Learning (ML) techniques for microarchitectural pattern analysis captured through Hardware Performance Counters (HPCs) to detect the SCAs with latency by order of ranging magnitude from milliseconds to seconds. For instance, the work in [4] proposes to detect the SCAs with the usage of both victim and attack applications' HPCs traces. Then based on the obtained HPCs, the correlation between the HPC events of victims' and attacks' traces will be evaluated. Similarly, in [24] the authors present CloudRadar which aims at detecting cross-VM side-channel attacks by making use of HPC patterns. The existing works on detection of SCAs based on low-level microarchitectural features have considered capturing both victim applications (cryptographic application, e.g. RSA, AES and etc.) and attack applications' microarchitectural features to detect the existence of the attacks. However, the attack HPCs data can be easily contaminated that could mislead the deployed ML-based SCA detection mechanism. In addition, the prior studies have explored the suitability of a limited number of ML algorithms in detecting microarchitectural SCAs.

In this work, we perform a comprehensive assessment of various machine learning-based countermeasures for real-time side-channel attack detection using HPC-based microarchitectural features. To eliminate the impact of missing attack profiling data or manipulation in the attack applications, this work proposes to detect SCA at real-time using the minimal number of HPC features (only 4 features). The deployed ML classifiers detect SCAs based on differentiating HPCs data of only the victim applications under two victim under attack and victim under no attack conditions. We further explore the HPCs monitoring overhead when microarchitectural features are sampled at different intervals to find out the appropriate sampling interval for SCA detection. For the purpose of thorough analysis, various types of ML classifiers are implemented and precisely comparedacross different evaluation metrics including detection accuracy, F-measure, ROC curve analysis, and computational latency to identify the most accurate and cost-efficient ML classifiers for real-time microarchitectural SCAs detection.

The remainder of this paper is organized as follows. The background and motivations are described in Section 2. The proposed framework and experimental setup details are discussed in Section 3. Section 4 presents the experimental results and provides a comprehensive analysis of different ML-based SCA detectors across various metrics. Finally, Section 5 presents the conclusion of this study.

2 BACKGROUND AND MOTIVATIONS

In this section, we highlight the background, relate works, and key motivations for proposing a comprehensive analysis for detecting side-channel attacks using victims' low-level hardware features.

2.1 Microarchitectural Side-channel Attacks

The emergence of different hardware components such as cache memory, branch predictor, etc. to enhance the performance of the modern microprocessors, have led to exposure of new hardware vulnerabilities in the systems. This makes a unique opportunity for the attackers to exploit such vulnerabilities by deducing sensitive information which results in microarchitectural side-channel attacks.

2.1.1 *Flush+Reload.*

The researches in [3, 22] exploits the vulnerability of page de-duplication technique by monitoring the memory access lines in the shared pages. This attack targets the Last-Level Cache in the CPU and flushes out victim applications' data in the cache and waits for the victim application to execute. After flushing the cache, the attacker tries to access the data and measures the accessing time (latency). Shorter accessing time denotes that the victim application has accessed the data; otherwise, it has not been accessed.

2.1.2 *Flush+Flush.*

This SCA relies on the execution time of the flush instruction [7]. Unlike prior attacks, Flush-Flush does not make any memory accesses, nor it relies on the access latency of the data. The execution time of flush instruction depends on whether the data is stored in the cache. Flush-Flush uses the execution time of the subsequent flush instruction following the victim application's execution. The large execution time of the flush instruction is indicative of the fact that, the corresponding data was brought to the cache and later accessed by the victim application.

2.1.3 *Prime+Probe.*

Without the memory de-duplication restriction, Prime+Probe [11] could be applied to more systems. This type of SCA consists of two different stages: Prime and Probe. In the Prime stage, the attacker builds the eviction sets which are group cache sets causing potential conflict with victim applications and then fills the cache with the eviction sets. Next, the attacker waits for the execution of the victim application and then re-accesses the eviction sets (Probe stage). If the accessing time is long enough, it means the victim application has accessed the data; otherwise, the victim application does not access it.

2.2 Related Work

The detection work in [1] monitors HPCs trace of both victim and attack processes and compare the effectiveness of three ML classifiers: neural network, decision tree C4.5, and K nearest neighbors. The work in [13] proposes a detection system containing one analytic server and one or more monitored computing devices to detect SCAs, including Spectre and Meltdown. The analytic server receives HPCs data from monitored devices and identifies suspicious core activity. Once detected, application level monitor will be deployed on the computing devices and take corrective actions as soon as finding suspicious application activity. Recent work [21] uses cache latency to build cache occupancy of victims and attacks. Based on the cache occupancy relation of the two processes, SCAs can be deduced.

Chiappetta et al. [4] collected HPC features for building the ML classifiers and compare three different attacks scenarios including finding a correlation between victims and attacks, building supervised machine learning models based on HPCs from victims and attacks, and detecting anomalies by validating attack HPCs as samples and other processes as outliers. Similarly, in [24] the authors presents CloudRadar which aims at detecting cross-VM side-channel attacks by deploying HPC patterns. The research in [13] proposes a detection system containing one analytic server and one or more monitored computing devices to detect SCAs including Spectre and Meltdown. The analytic server receives HPCs data from monitored devices and identifies suspicious core activity. Once detected, application level monitor is deployed and take corrective actions. The work in [10] proposes an online detection of Spectre by monitoring microarchitectural features using time series classification.

2.3 Detection based on Victims' HPCs Data

Current SCAs intentionally cause influence on victim applications' cache or branch predictor by flushing/priming cache, mistraining branch predictors and then observe accessing time of the cache sets, which changes caching victims' data and microarchitectural behaviors of victim applications [25]. This also provides the opportunity of detecting SCAs by observing the alteration in microarchitectural behaviors. Furthermore, our experimental results as shown in Figure 1 indicate that there exists a clear difference between the behavior of victim under no attack (VNA) and victim under attack (VA). In this motivational case study, the HPC traces of L1 HIT for the tested victim application (RSA) under no attack (RSA) and under L3 Flush Reload attack (RSA with FR) are illustrated. It can be observed that the L1 HIT of VA shows a significantly different trend compared to that of VNA. This observation clearly highlights the effectiveness of using HPCs data of only victim applications (excluding the impact of attack applications' HPCs) for detecting the behavior of SCAs.

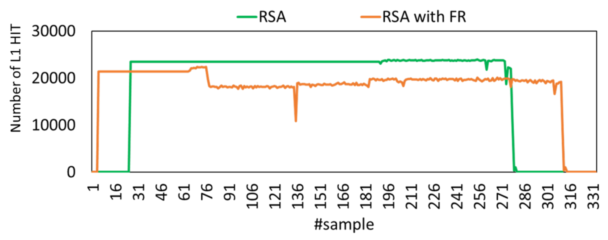

Figure 1: L1 HIT of RSA and RSA under Flush Reload attack

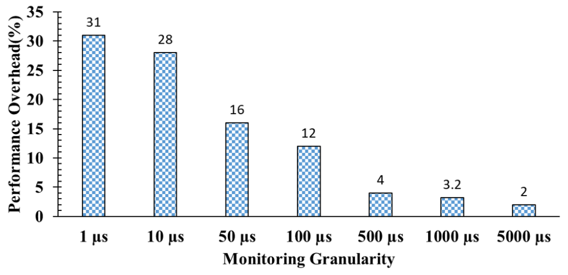

Figure 2: Performance overhead with various monitoring granularities

2.4 HPCs Monitoring Overhead

Hardware performance counters are a set of special-purpose registers built in modern microprocessors to capture the count of hardware-related events which have been extensively used to predict the power, performance, malware detection, and energy efficiency of computing systems [15, 17, 18]. Recently, there has been a number of work on hardware-based side-channel attacks detection using HPCs information [2, 14, 20, 24]. The detectors are based on machine learning classifiers which are trained with processor HPCs data captured at run-time to appropriately represent application behavior. However, these works ignored the performance overhead caused by HPCs monitoring and focused on limited machine learning classifiers. In this section, we examine the performance overhead caused by HPCs monitoring tool of both victim and benign applications under various monitoring granularities. As shown in Figure 2, the x-axis represents applied monitoring granularity ranging from 1 μs to 5000 μs, the primary y-axis represents the execution time of victim applications and the second y-axis represents performance overhead under different monitoring granularities. Execution time under no HPCs monitoring is used to obtain performance overhead percentage. It is observed that, generally, the smaller the monitoring granularity, the larger the performance overhead. When the monitoring scale is 1 μs, performance overhead is at its highest value reaching to 30%. Due to the large difference of monitoring overheads, it is important to determine a proper level of monitoring granularity to balance the detection performance and HPCs monitoring costs.

Table 1: List of HPC events collected for SCAs detection

L1 HIT	L1 MISSES
L2 HIT	L2 MISSES
L3 HIT	L3 MISSES
All BRANCHES RETIRED	BRANCHES MISPREDICTED
BR_NONTAKEN_CONDITIONAL	BR_TAKEN_CONDITIONAL
TAKEN_INDIRECT_NEAR_CALL	UOPS_RETIRED.ALL
INST_RETIRED.ANY	DTLB_LOAD_MISSES
DTLB_STORE_MISSES	ITLB_MISSES

3 PROPOSED METHODOLOGY

In this section, we first present details of the experimental setup and configurations. And then the evaluation methodology shown in Figure 3 will be introduced. As shown, the proposed approach is comprised of different steps such as data collection, training phase, cross validation phase, and testing phase. First, HPC data will be collected within a) isolated scenario, and b) non-isolated scenario when sampling interval is set to different value including 50 μs, 100μs, 500μs, and 1000μs, respectively. The "isolated" environment refers to the case that a computer only processes victim applications; whereas the "non-isolated" environment denotes that a computer system processes victim applications on one core while benign applications are being executed on the rest of the processing cores.

3.1 Experimental Setup and Data Collection

In this work, all experiments are conducted on an Intel I5-3470 desktop with 4 cores, 8GB DRAM, and three-level cache system. Victim applications and side-channel attacks (Flush+Reload, Flush+Flush, and Prime+Probe) are selected from Mastik [22]. Furthermore, MiBench [8] benchmark suite is used to represent benign applications. We further propose using a customized tool to collect hardware performance counters based on Model-Specific Registers (MSRs). The proposed customized monitoring tool collects HPCs per processor at microsecond scale with privileged access to avoid HPCs contamination from other processes addressing the overcounting challenges presented in a recent study [5]. Based on the behavior and functionality of studied SCAs, 16 HPC features are considered in this work for further analysis as listed in Table 1. These hardware performance counters data are collected using the four available HPC registers in the experimented I5 processor at every sampling interval (50/100/500/1000 μs). Next, both VA and VNA HPC data from each same sampling intervals are merged together to create the final dataset for the corresponding sampling interval.

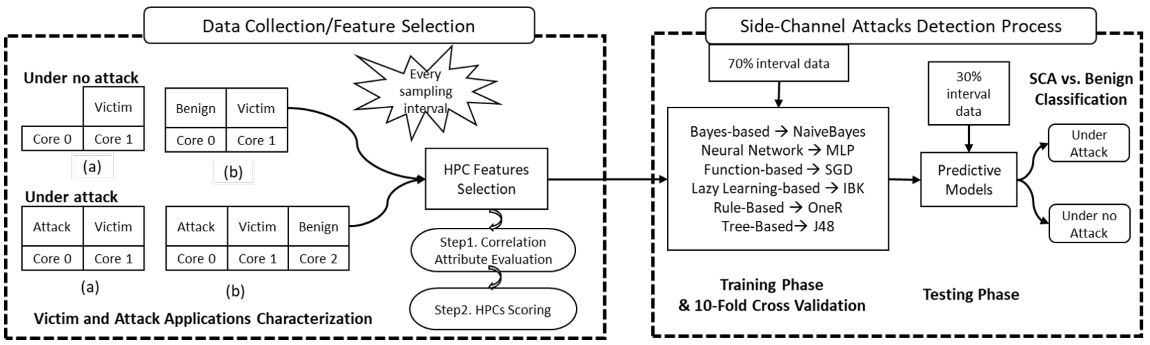

Figure 3: Overview of proposed methodology for comprehensive evaluation of ML countermeasures for SCAs detection

Table 2: Evaluated ML classifiers for microarchitectural SCAs detection

ML Category	Notation	Selected Classifier
Bayesian Network	Algorithms that use Bayes Theorem in some core way, like Naive Bayes.	NaiveBayes
Neural Network	Series of algorithms that attempt to recognize and mimic the human brain operations.	MultilayerPerceptron (MLP)
Support Vector Machine	Linear model for classification and regression problems.	SGD
Lazy	Algorithms that use lazy learning, like k-Nearest Neighbors.	IBK
Rules	Algorithms that use rules, like One Rule.	OneR
Trees	Algorithms that use decision trees, like Random Forest.	J48

3.2 HPCs Feature Reduction

Given the limited number of HPCs available in modern microprocessors (only 4 HPCs on tested Intel I5-3470) to be collected at one time simultaneously, it is necessary to identify the most important HPCs for SCAs detection. Furthermore, incorporating irrelevant features would lead to lower accuracy and performance for the classifiers. Hence, it is crucial to perform an effective feature reduction of collected data to alleviate unnecessary computational overheads and determine the most prominent low-level features [15, 16]. In order to detect the SCAs at real-time with minimal overhead, we intend to identify a minimal set of critical HPCs that are feasible to collect even on low-end processors with small number of HPCs in a single run. Since experiment platform has 4 HPCs registers, therefore, 4 HPC features is selected representing the most important features for classification. For HPCs reduction, we employ Correlation Attribute Evaluation (*CorrelationAttributeEval* in Weka) with its default settings to calculate the Pearson correlation between attributes (HPC features) and class (VA and VNA conditions). Correlation attribute evaluation algorithm calculates the Pearson correlation coefficient between each attribute and class to identify the most prominent features for SCAs detection depending on the selected sampling interval.

3.3 ML Classifiers Implementation

Table 2 describes the ML classifiers evaluated in this work that are selected from five different categories. These ML classifiers include NaiveBayes, Multi-Layer Perceptron (MLP), SGD, IBK, OneR, and J48. The rationale for selecting these machine learning models is that they are from different branches of ML including Bayesian network-based, neural network, support vector machine, lazy learning-based, rule-based, and tree-based techniques covering a diverse range of learning algorithms which are inclusive to model both linear and nonlinear problems. In addition, the prediction model produced by these learning algorithms can be a binary classification model which is compatible with the SCAs detection problem in our work. Furthermore, Weka data mining tool is deployed for implementing the machine learning classifiers. To validate each of the utilized ML classifiers, first a standard 70%-30% dataset split for training and testing is followed. Next, for the percentage split testing 70% of the randomized data is used for training the classifiers and the rest of 30% is used for testing evaluation. In addition, a k-fold (k=10) cross-validation is conducted using only the training dataset.

4 RESULTS EVALUATION

In this section, we evaluate the effectiveness of ML-based SCAs detectors based on detection accuracy, F-measure, classification robustness, and incurred costs (monitoring and computation).

4.1 Detection Accuracy vs. Sampling Granularity

As mentioned in Section 2.4, the performance overhead of HPCs monitoring ranges from less than 5% to around 30% depending on the selected monitoring granularity. Due to space limitation, here we show the influence of sampling granularity with 50μs to 5000μs. As shown in Figure 4, detection accuracy of the six different ML classifiers for both cross validation and percentage split testing accuracy results ranges from 65% to 93% when HPCs are collected at every 50 μs. It is observed that increasing the sampling time interval from 50 μs to 100 μs significantly improves the accuracy for both cross validation and testing by up to 20% (NaiveBayes). Another observation is that increasing sampling interval beyond 100 μs does not have considerable influence on accuracy of classifiers. We believe that it is because using larger sampling interval (>50μs) could help to alleviate some levels of potential noises which results in improvement of detection accuracy. As demonstrated before, the

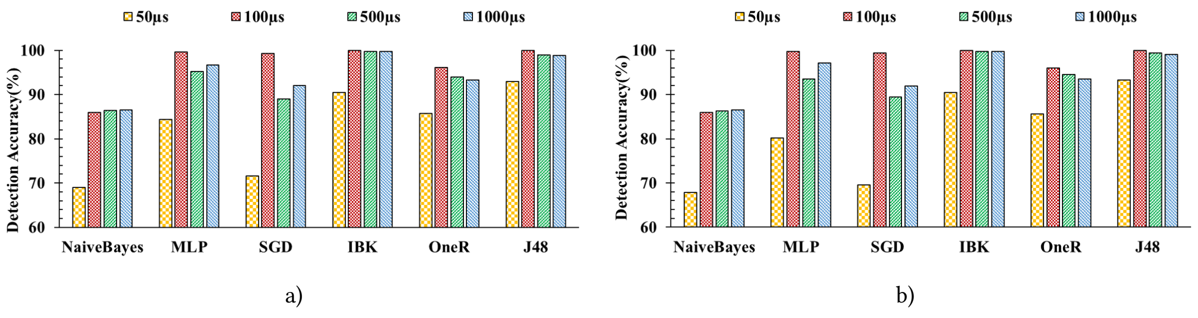

Figure 4: Prediction accuracy comparison with different sampling intervals (50/100/500/1000 μs): a) cross validation accuracy from six classifiers; b) testing accuracy from six classifiers.

lesser sampling interval results in the higher performance overhead. From 500 μs to higher sampling intervals, performance overhead remains stable and less than 5%. Furthermore, the real-time SCAs detector could spend less amount of time to receive the HPC data which is leading to a faster SCAs detection process. As a result, it can be concluded that using 500 μs provides a balanced trade-offs between detection accuracy, performance overhead, and data collection latency. Following, we will further demonstrate the robustness (ROC curve), F-measure and testing latency with 500 μs sampling interval for the all of the implemented ML-based detectors.

Table 3: F-measure of various classifiers

Classifiers	NavieBayes	MLP	SGD	IBK	OneR	J48
F-measure	0.862	0.934	0.894	0.998	0.945	0.993

4.2 F-measure

F-measure is interpreted as a weighted average of the precision (p) and recall (r) which is formulated as $\frac{2\times(p\times r)}{p+r}$. 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. Table 3 presents the F-measure results of all implemented ML classifiers. 5It can be observed that NaiveBayes has lowest F-measure, indicating that it is the least effective classifier for SCAs detection, whereas IBK and J48 have quite similar F-measure values (0.998 and 0.993) delivering the highest detection rate in terms of F-measure among all classifiers. It is also notable that MLP is more effective compared to SGD classifier in detecting SCAs.

4.3 Classification Robustness

Receiver Operating Characteristics (ROC) Curve is produced by plotting the fraction of true positives rate versus the fraction of false positives for a binary classifier. The best possible classifier would thus yield a point in the upper left corner or coordinate (0,1) of the ROC space, representing 0% false positives and 100% true positives. Area under the ROC Curve (AUC) metric. The AUC corresponds to the probability of correctly identifying "under attack" and "under no attack" and robustness is referred to how well the

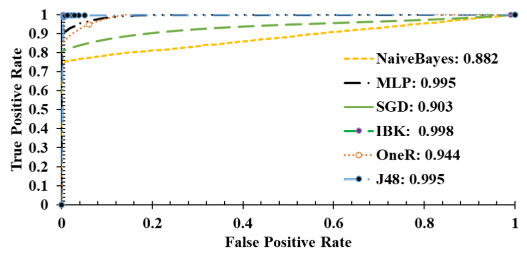

Figure 5: ROC Curve and AUC value of various classifiers

classifier distinguishes between the two classes, for all possible threshold values. Higher AUC indicates better robustness for ML classifiers. Figure 5 depicts the ROC Curve of various ML classifiers with corresponding AUC values. It can be observed that similar to F-measure, it can be observed in Figure 5 that the NaiveBayes algorithm performs the worst in terms of ROC Curve, having the biggest distance to point (0,1). The IBK and J48classifiers are closer to the coordinate (0,1), indicating higher true positive rate and less false positive rate compared with other four classifiers evaluated in this work. Though MLP has lower F-measure compared to IBK and J48, the ROC curve and AUC value is similar to the one in J48, indicating that the mispredicted instances are evenly distributed between "under attack" and "under no attack" classes.

Table 4: Classifiers latency

Classifiers	NaiveBayes	MLP	SGD	IBK	OneR	J48
Latency(μs)	212	252	237	312	235	228

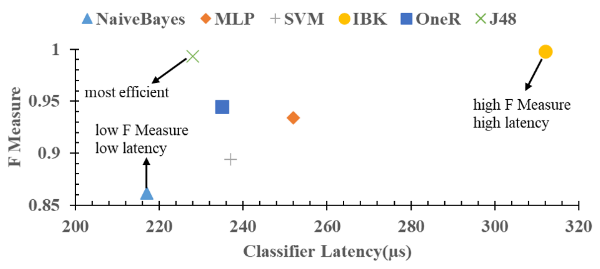

Figure 6: Efficiency comparison among various classifiers

4.4 Efficiency Analysis: F-measure vs. Latency

Table 4 presents the classification latency of various models, ranging from 212 μs to 312 μs. Compared to other ML classifiers, IBK has highest latency, indicating more computation and resource utilization. While J48 showed to have a similar F-measure as IBK, it has much lower latency (228 μs). Lastly, to accordingly account for both performance rate and cost of ML classifiers, in Figure 6 we compare detection rate over a computational latency (F-measure/Latency) for various ML classifiers. We use F-measure over latency to identify the SCA detectors that require small cost and yet can detect the maliciousness of program with high accuracy and performance. A classifier with a higher ratio is considered a more efficient detector than the classifier with lower ratio. As shown in Figure 6, a clear trade-off is seen between detection rate and latency achievable for real-time hardware-assisted SCAs detection. The ML classifiers such as IBK achieves high detection rate, but also higher computational overhead. The techniques such as NaiveBayes, MLP, SGD, IBK, OneR, and J48 show relatively smaller timing costs with high SCAs F-measure. For highly resource-constrained embedded systems, techniques such as J48 provide smallest computational overhead, while achieving an F-measure of close to 0.993 on average. Clearly, the results show trade-offs between F-measure and latency. Therefore, it is important to compare ML classifiers for effective SCAs detection by taking all these parameters into consideration.

5 CONCLUSION

In this work, we propose a comprehensive analysis of various Machine Learning (Ml) classifiers for detecting microarchitectural side-channel attacks (SCAs) at real-time using the processor's Hardware Performance Counters (HPCs) information. The proposed methodology further presents the performance overhead of HPCs monitoring and evaluates the most suitable sampling interval to balance performance overhead and detection rate. Moreover, it addresses the challenge of the lack of attacks applications' HPCs data by analyzing the difference between Victim under Attack (VA) and Victim Under No Attack (VNA) conditions. Our thorough analysis indicates that HPCs data of VNA and VA show significantly different behavior providing the opportunity to detect SCAs with only victim applications' HPCs data. We use HPCs importance evaluation with Correlation Attribute Evaluation algorithm to identify the most prominent HPC features suitable for real-time SCA detection. For an in-depth analysis, various machine learning classifiers are implemented and precisely compared in terms of detection accuracy, F-measure, robustness (Area Under the ROC Curve), and computational latency to determine the most efficient ML classifiers for real-time microarchitectural SCAs detection. The results of this research highlights important design principles and trade-offs analysis in implementing accurate and cost-efficient machine learning-based countermeasures for securing modern processor architectures against emerging microarchitectural side-channel attacks.

6 ACKNOWLEDGMENT

This research was supported in part by DARPA SSITH program under the award number 97774952.

REFERENCES

[1] Allaf, Z., and et.all. A comparison study on flush+ reload and prime+ probe attacks on aes using machine learning approaches. In *UK Workshop on Computational Intelligence* (2017), Springer.
[2] Briongos, S., Irazoqui, G., Malagón, P., and Eisenbarth, T. Cacheshield: Detecting cache attacks through self-observation. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy* (2018), ACM, pp. 224–235.
[3] Chiappetta, M., Savas, E., and Yilmaz, C. Xlate: https://www.vusec.net/projects/xlate/.
[4] Chiappetta, M., Savas, E., and Yilmaz, C. Real time detection of cache-based side-channel attacks using hardware performance counters. *Applied Soft Computing 49* (2016), 1162–1174.
[5] Das, S., Werner, J., Antonakakis, M., Polychronakis, M., and Monrose, F. Sok: The challenges, pitfalls, and perils of using hardware performance counters for security. In *2019 IEEE Symposium on Security and Privacy (SP)* (2019), IEEE, pp. 20–38.
[6] Depoix, J., and Altmeyer, P. Detecting spectre attacks by identifying cache side-channel attacks using machine learning. *Advanced Microkernel Operating Systems* (2018), 75.
[7] Gruss, D., Maurice, C., Wagner, K., and Mangard, S. Flush+ flush: a fast and stealthy cache attack. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment* (2016), Springer, pp. 279–299.
[8] Guthaus, M. R., Ringenberg, J. S., Ernst, D., Austin, T. M., Mudge, T., and Brown, R. B. Mibench: A free, commercially representative embedded benchmark suite. In *Fourth WWC-4* (2001), IEEE.
[9] Kocher, P., and et.all. Spectre attacks: Exploiting speculative execution. *arXiv preprint arXiv:1801.01203* (2018).
[10] Li, C., and Gaudiot, J.-L. Online detection of spectre attacks using microarchitectural traces from performance counters. In *2018 30th SBAC-PAD*, IEEE.
[11] Liu, F., Yarom, Y., Ge, Q., Heiser, G., and Lee, R. B. Last-level cache side-channel attacks are practical. In *Security and Privacy (SP), 2015 IEEE Symposium on* (2015), IEEE, pp. 605–622.
[12] Mirzaeyan, A., Patooghy, A., and Ali, M. F. A novel countermeasure against fault injection attacks for aes-based cryptosystems. In *2016 24th Iranian Conference on Electrical Engineering (ICEE)* (2016), pp. 1148–1153.
[13] Prada, I., Igual, F. D., and Olcoz, K. Detecting time-fragmented cache attacks against aes using performance monitoring counters. *arXiv preprint arXiv:1904.11268* (2019).
[14] Sabbagh, M., and et.all. Scadet: A side-channel attack detection tool for tracking prime-probe. In *2018 ICCAD* (2018), IEEE.
[15] Sayadi, H., and et al. Machine learning-based approaches for energy-efficiency prediction and scheduling in composite cores architectures. In *ICCD'17* (Nov 2017), pp. 129–136.
[16] Sayadi, H., and et.all. 2smart: A two-stage machine learning-based approach for run-time specialized hardware-assisted malware detection. In *2019 DATE*, IEEE.
[17] 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 *2018 55th DAC*, IEEE.
[18] Wang, H., Rafatirad, S., and Homayoun, H. A+ tuning: Architecture+ application auto-tuning for in-memory data-processing frameworks. In *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)* (2019), IEEE, pp. 163–166.
[19] Wang, H., Sayadi, H., Mohsenin, T., Zhao, L., Sasan, A., Rafatirad, S., and Homayoun, H. Mitigating cache-based side-channel attacks through randomization: A comprehensive system and architecture level analysis. In *DATE* (2020), IEEE.
[20] Wang, H., Sayadi, H., Rafatirad, S., Sasan, A., and Homayoun, H. Scarf: Detecting side-channel attacks at real-time using low-level hardware features. In *IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS'20)* (2020), IEEE.
[21] Yao, F., Fang, H., Doroslovacki, M., and Venkataramani, G. Towards a better indicator for cache timing channels. *arXiv preprint arXiv:1902.04711* (2019).
[22] Yarom, Y. Mastik: A micro-architectural side-channel toolkit. *Retrieved from School of Computer Science Adelaide: http://cs. adelaide. edu. au/˜ yval/Mastik* (2016).
[23] Yarom, Y., and et.all. Flush+ reload: A high resolution, low noise, l3 cache side-channel attack. In *USENIX Security Symposium* (2014).
[24] Zhang, T., and et.all. Cloudradar: A real-time side-channel attack detection system in clouds. In *RAID* (2016), Springer.
[25] Zhang, T., and Lee, R. B. Secure cache modeling for measuring side-channel leakage. *Technical Report, Princeton University* (2014).