SCARF: Detecting Side-Channel Attacks at Real-time using Low-level Hardware Features

Han Wang[1], Hossein Sayadi[2], Setareh Rafatirad[3], Avesta Sasan[3], and Houman Homayoun[1]
[1]University of California, Davis, CA, USA
[2]California State University, Long Beach, CA, USA
[3] George Mason University, Fairfax, VA, USA
[1]{hjlwang,hhomayoun}@ucdavis.edu, [2]{hossein.sayadi}@csulb.edu,[3]{srafatir,asasan}@gmu.edu

***Abstract*—Side-Channel Attacks (SCAs) are powerful attacks compromising the security of modern computer systems by exploiting hardware vulnerabilities. Prior studies on detection of SCAs based on low-level microarchitectural features captured from processors' hardware performance counter (HPC) registers have considered collecting hardware events of both victim applications (cryptographic application, e.g. RSA, AES and etc.) and attack applications. However, in such techniques the attack HPCs data can be easily manipulated and/or corrupted resulting in misleading the SCA detection mechanism. Furthermore, the prior works have explored the suitability of a limited number of Machine Learning (ML) algorithms in detecting SCAs without examining the instance level false alarm rate that as we show in this work is a more important evaluation metric for SCA detection techniques. In response, in this paper, we propose *SCARF*, a machine learning-based real-time side-channel attack detection methodology using low-level hardware features. To this aim, we first only monitor the victim applications' behavior using the HPC features and analyze the captured low-level traces of the victim applications under no attack and attack conditions to avoid manipulation of attackers' HPCs. Next, a wide range of ML classifiers with customized HPC features are implemented to determine the most effective ML technique for detecting SCAs at real-time, while improving accuracy and reducing instance-level false alarm rate of ML-based SCA detectors. Lastly, the False Alarm Minimization (FAM) technique is proposed to further reduce the instance level false positive rate of the ML-based SCA detectors. The experimental results indicate that the *SCARF* methodology can obtain up to 100% attack detection accuracy with 0% instance level false alarm rate for detecting SCAs.**

I. Introduction

Side-channel attacks (SCAs) primarily target inferring sensitive and confidential information from a computer system through measurement and analysis of physical parameters [1], [16], [17]. 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 [10], [19]. As a result, 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.

Prior works on side-channel attack detection such as [5], [7], [20] propose the use of 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 [5] 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 [20] the authors present CloudRadar which aims at detecting cross-VM side-channel attacks by making use of HPC patterns. Undoubtedly, the prior detection works have made some progress in detecting the SCAs. However, they fall short in addressing the challenges defined above as well as several drawbacks, as demonstrated below.

❶ **Lack of Robustness:** Our comprehensive analysis shows that the majority of previous works on side-channel attacks detection jointly correlate the HPCs traces of victim and attack applications [5], [20]. However, recent studies [6], [22] have demonstrated that current HPC features monitoring methods suffer from the overcounting issue that creates the opportunity for attackers to manipulate HPCs data by slightly changing SCA applications. Hence, current detection techniques relying on the HPCs data of attack applications are facing with significant security threats.

❷ **Limited Machine Learning Classifiers:** A wide range of classification and anomaly detection techniques are developed by applying Machine Learning (ML) techniques. However, existing works in particular on SCA detection have primarily focused on one or a few ML techniques for the purpose of attack detection and classification [5], [20]. Such an analysis leaves a void in terms of performance of attack detection, as various ML classifiers yield different performance in detecting various types of attacks [15].

❸ **High False Alarm Rate:** Prior studies on real-time HPC-based SCAs detection have neglected to examine the instance level (a complete temporal sequence of victim applications' HPCs) false positives of HPC data and have only evaluated the SCAs detectors based on the interval level (a sub-sequence of victim applications HPCs) false positive rate [5], [20]. However, SCA detection based on the capturing intervals of HPCs data is biased to "under attack" conditions. The Victim under No Attack (VNA) requires all the captured HPCs intervals to be classified correctly by the machine learning-based SCA detector to achieve a correct prediction while Victim under Attack (VA) requires only one interval classified correctly to achieve a correct prediction.

To address the aforementioned challenges, in this work we propose *SCARF*, a unified and accurate machine learning-based real-time side-channel attack detection methodology using low-level hardware features. To this end, we first demon-

strate the validation of detecting side-channel attacks based on the victim applications HPCs data to avoid manipulation of the attacker's HPC by executing the victim applications on an isolated processing core. Under such isolated circumstances, we find that benign applications have significantly less impact on victim applications' HPCs compared to side-channel attack applications. Next, we present the problem of instance level false positive rate and demonstrate the importance of extracting customized features from HPC traces intervals in order to achieve a higher detection accuracy and lower instance level false positive rate for ML-based SCA detectors. Then, the proposed real-time SCA detection methodology employs the customized features based ML classifiers by using only victims applications' microarchitectural information to enhance the SCAs detection accuracy and minimize the instance false positives rate while avoiding the corrupted, manipulated or missing attackers' HPCs information. The main contributions of this work can be summarized as below:

- To eliminate the influence of missing attack profiling data or manipulation in the attack applications codes, this work proposes *SCARF* to detect SCA attacks at real-time using the minimal number of HPC features (only 4 features). The proposed approach detect SCAs based on differentiating HPCs data of only the victim applications under two conditions: 1) Victim under Attack (VA), and 2) Victim under No Attack (VNA).
- Various ML classification algorithms are explored to find the most accurate classifier for detecting side-channel attacks at real-time.
- The proposed ML-based detectors are trained using a customized set of HPC features to further improve the detection accuracy while lowering the false alarm rate.
- The False Alarm Minimization (FAM) method is proposed to reduce the instance level false positive rate with limited latency.

II. Motivation and Background

A. Motivational Case Studies

In this section, we highlight key motivations behind proposing *SCARF* framework for detecting side-channel attacks using low-level hardware features.

1) Detection based on Victim Applications' HPCs Data:
- *Unreliable Attackers' HPCs:* Prior studies on SCAs detection have mostly focused on profiling both victim and attack applications to collect hardware performance counters data for detecting whether an attack occurs or not [5], [7], [20]. Nevertheless, a recent work [22] presents the problem of detecting attackers by classifying attackers and benign applications based on HPC information with the aid of ML techniques. The work in [6] further points out the non-deterministic and over-counting problems of instructions associated with HPCs information, in which the attackers can intentionally modify instructions slightly and manipulate the counters, hence thwarting such detectors.
- *SCAs Design Principle:* Current SCAs intentionally cause influence on victim applications' cache or branch predictor

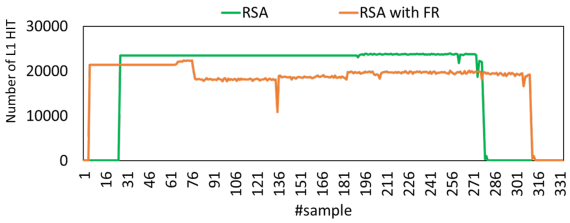

Fig. 1. L1 HIT of RSA and RSA under Flush Reload attack

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 [21]. 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 VNA and 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.

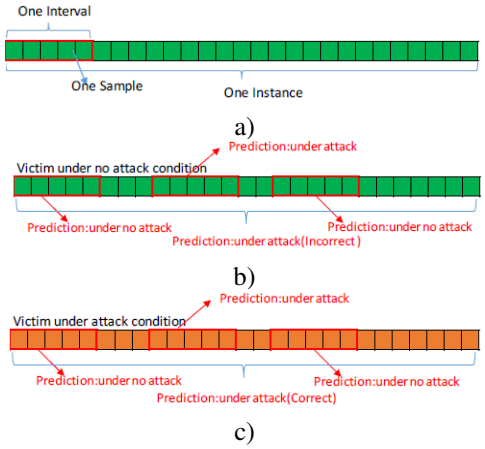

Fig. 2. False Alarm Problem: a) Concept; b) VNA condition; c) VA condition

2) False Alarm Problem: As depicted in Figure 2-(a), each run of a victim application is called an instance. For the purpose of real-time SCAs detection, a certain window size is used to decide the number of samples of each interval. Each instance could contain multiple intervals. In addition, Figure 2-(b) shows a VNA instance divided into multiple intervals. In such cases, even if only one interval is predicted as VA by the ML-based detection technique, the whole instance will be classified incorrectly as VA. At the same time, Figure 2-(c) illustrates that the VA instance has two intervals classified as VNA and one interval classified as VA, the whole instance is still correctly classified as VA. Hence, even for prior works [5], [20] achieving high detection accuracy with low false positives, it is still hard to say that they can achieve a low instance level false alarm rate. To distinguish false positives

TABLE I. FALSE POSITIVE AND FALSE NEGATIVE EVALUATION (TRUE: VNA; FALSE: VA)

	Predicted True(interval)	Predicted False(interval)	Predicted True(instance)	Predicted False(instance)
Actual True	True Positive	**False Positive**	True Positive	**False Alarm**
Actual False	**False Negative**	True Negative	**Missed Alarm**	True Alarm

of interval level and instance level, we deploys false alarm and missed alarm to represent false positive and false negative of instance level in the following sections as shown in Table I.

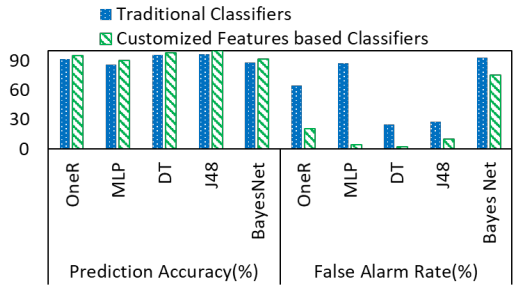

Fig. 3. Traditional and customized features based classifiers comparison (datasets collected based on Section III-A)

3) Customized Features based ML Classifiers: Prior works capture the sum of HPCs value for a certain time period as features and employ traditional ML classification methods, achieving high prediction accuracy. They can cause a high false alarm rate as mentioned in Section II-A2. Hence, customized features based ML classifiers which extract more features, like min, max, stdev, and sum of an interval are required to further boost the prediction accuracy and reduce the false alarm rate. Prediction accuracy and false alarm rate of traditional and customized features based classifiers are plotted in Figure 3. It can be seen that customized features based ML classifiers outperform traditional ones by around 4% prediction accuracy. Furthermore, the false positive rate drops from 87.2% to 4.7% for MLP when applying customized features based classifiers. It can be concluded that extracting more features from HPCs time-series sequences can help ease the "under attack" bias mentioned in Section II-A2.

B. Relevant Works

The work in [5] proposes an HPC monitoring model in order to detect the SCAs of both victims and attacks applications. Further, based on the obtained HPCs, the correlation between the HPC events of victim and attacks' traces. Similarly, in [20] the authors presents CloudRadar which aims at detecting cross-VM side-channel attacks by deploying HPC patterns. The research in [12] 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. The work in [11] proposes an online detection of Spectre attack by monitoring microarchitectural features using time series classification. However, it only targets Spectre attack and is not able to provide more comprehensive protection from SCAs. In addition, all these works can be bypassed when SCAs manipulate HPCs values according to HPC monitoring challenges discussed in recent work [6].

TABLE II. THE EXPERIMENTED VICTIM AND ATTACK APPLICATIONS

Victim	Attack	Source
RSA	L3 Flush Reload	Masitk [18]
	L1 Prime Probe	Masitk
AES	Flush Reload	Xlate [4]
	L3 Flush Flush	Xlate
victim_function	Spectre	Spectre [3]

TABLE III. THE COLLECTED HPC FEATURES AND THEIR RANKING

Ranking	HPC Name	Ranking	HPC Name
1	L1 HIT	9	L1 MISSES
2	UOPS_RETIRED	10	BRANCHES MISPREDICTED
3	BR_NONTAKEN_CONDICTIONAL	11	L2 HIT
4	ALL BRANCHES RETIRED	12	TAKEN_INDIRECT_NEAR_CALL
5	INST_RETIRED_ANY	13	L3 HIT
6	L2 MISSES	14	ITLB_MISSES
7	BR_TAKEN_CONDITIONAL	15	DTLB_STORE_MISSES
8	L3 MISSES	16	DTLB_LOAD_MISSES

III. PROPOSED METHODOLOGY

In this section, we first present details of the experimental setup and configurations. And then the proposed *SCARF* methodology shown in Figure 4 will be introduced. As shown, *SCARF* is comprised of different steps such as data collection, feature reduction, training phase, testing phase, and false alarm minimization method. First, for feature extraction the "under no attack" and "under attack" HPC data will be collected within a) isolated scenario, and b) non-isolated scenario. 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 cores. Then customized features are extracted and the data will be used to train various classifiers. Next, the trained models will be employed in the testing phase and false alarm minimization technique further assists in reducing false alarm rate.

A. 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 are selected from Mastik [18] and Xlate [4]. Furthermore, MiBench [9] benchmark suite is used to represent benign applications. In this work, we 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 [6]. Based on the behavior and functionality of studied SCAs, 16 HPC features are considered in this work for further analysis as listed in Table III. These hardware performance counters data are collected using the four available HPC registers in the experimented I5 processor at every 50 microseconds. Each pair of a VNA and VA executes for 50

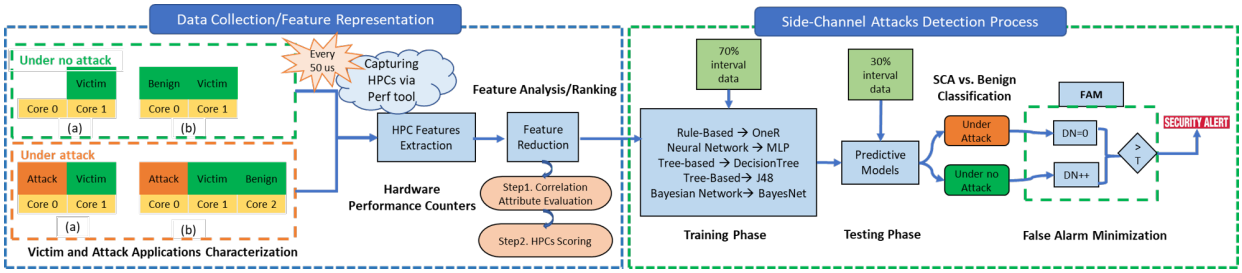

Fig. 4. Overview of *SCARF*, the proposed real-time SCAs detection methodology based on victim application HPCs

times. Next, both VA and VNA HPC data are merged together to create the final dataset. Furthermore, Weka data mining tool is deployed for implementing the machine learning classifiers. To validate each of the utilized ML classifiers, a standard 70%-30% non-biased dataset split for training and testing is followed in which 70% of the randomized data (known applications) is used for training the classifiers and the rest of 30% (unknown applications) is used for testing evaluation.

B. Customized Features based Classifiers

The proposed customized features based classification is comprised of three main steps: 1) feature extraction and representation; 2) HPCs selection due to a limited number of registers for effective real-time detection of the attacks; 3) training the ML classification algorithms.

1) Features Vector Extraction and Representation: For proposed classification, the raw data will be transformed from a time sub-sequence to a vector of features. In the first step of the transformation process, the raw data will be received from the monitoring module. The time-series sub-sequences' properties will be extracted which include statistics of distribution values (including max, min, StatAv and sum). In order to effectively determine the most prominent features for the purpose of real-time SCA detection, we deployed Greedy Forward Selection algorithm [2], [8] and we found that max, min, StatAv and sum contribute more to assisting in distinguishing the difference between "under no attack" and "under attack" conditions. Hence, the input for each transformation is $T = (t1, t2, ..., tm)$ where tm is a vector of HPCs values. Also, the outputs of transformation are a vector of actual HPC values i.e., L1HIT sum, L1 HIT max, etc.

2) HPCs Feature Reduction: Detecting side-channel attacks using ML models requires representing programs at low-level features which leads to a high-dimensional data processing involved large computational overheads and complexity. 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 [13], [14]. In order to detect the SCAs at real-time with minimal overhead in *SCARF*, 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. Therefore, a subset of HPC features is selected representing the most important features for classification. The selected features are then supplied to each ML-based SCA detector. The detector attempts to find a correlation between the feature values and the application behavior to predict the SCA.

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 classifying the VA and VNA conditions for different types of SCAs [15]. 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, as given below:

$$\rho(i) = \frac{cov(Z_i, C)}{\sqrt{var(Z_i)\ var(C)}} \quad i = 1, ..., 16 \qquad (1)$$

where ρ is the Pearson correlation coefficient. Z_i is the input dataset of event i ($i = 1, \ldots, 16$). C is the output dataset containing labels, i.e. "Under Attack" or "Under No Attack" in our case. The $cov(Z_i, C)$ measures the covariance between input data and output data. The $var(Z_i)$ and $var(C)$ measure variance of both input and output datasets, respectively. Next, the sum score of each HPC features (min, max, stdev, and sum in this work) will be calculated and HPCs will be ranked according to sum score as shown in Table III.

C. ML Classifiers Implementation

For the purpose of a thorough analysis of various types of ML classifiers, OneR, MLP (multilayer perceptron), DT (decision table), J48, and BayesNet ML algorithms are deployed as our final classification models. The rationale for selecting these machine learning models are: firstly, they are from different branches of ML: regression, neural network, decision tree, and rule-based techniques covering a diverse range of learning algorithms which are inclusive to model both linear and nonlinear problems; secondly, the prediction model produced by these learning algorithms can be a binary classification model which is compatible with the SCA detection problem in our work. As mentioned before, only four HPCs can be collected for most processors at once due to a limited number of registers for storing them. Hence, reducing the number pf HPCs required for ML models is important to eliminate the need of multiple runs. For this purpose, various number of HPCs from 16 to 4 (16, 12, 8 and 4 selected based on the ranking in Table III) are examined to evaluate the influence of reduced HPCs on classification accuracy and highlight the importance and motivation of using a lower number of HPCs (only 4) for effective real-time SCA detection in *SCARF*.

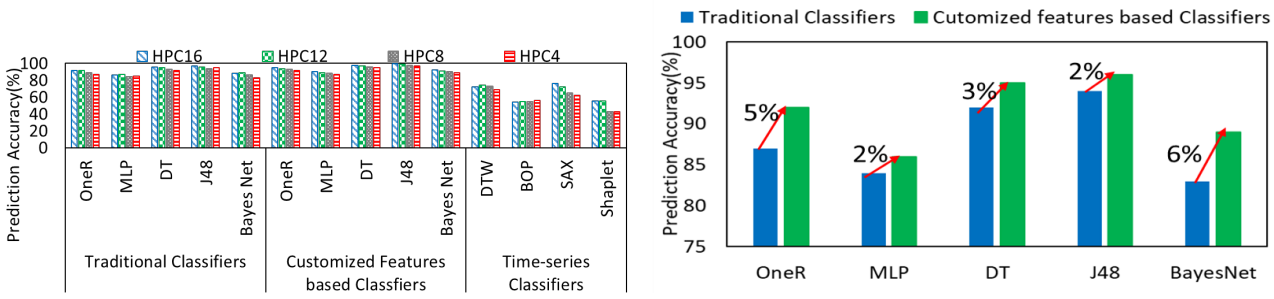

Fig. 5. Prediction accuracy comparison: a) prediction accuracy of proposed customized features based classifiers and the rest two type classifiers; b) zooming in prediction accuracy of traditional and customized features based classifiers

D. False Alarm Minimization (FAM)

As mentioned in Section II-A, previous real-time SCAs detection methods are biased to under attack category that result in increasing the false alarm rate. Hence, to address this challenge, we propose False Alarm Minimization (FAM) technique in which we delay the "under attack" detection decision until receiving a certain number of continuous intervals, delay number (DN), before reporting as "under attack", while minimizing detection latency.

To this aim, we assume false positives are evenly distributed among each instance, which results in highest false alarm rate with the same false positive rate. Following, the value setting of DN will be demonstrated. As mentioned above that evenly distributed false positives are leading to highest potential false alarm rate, we propose the method of setting DN value to ensure the highest potential false alarm rate with known false positive rate and the number of instance intervals which can be obtained after testing classification models. First, we suppose $DN = m$, the number of $intervals = n$, false positive $rate = s$ and acceptable false alarm rate is t. There are $n - m + 1$ possible cases of m consecutive intervals that are incorrectly identified as "under attack". Therefore, the false alarm rate can be calculated by $FAR = (n - m + 1) * (s\%)^m < t$. Since n, s and t are known, minimum DN value can be deduced according to the equation.

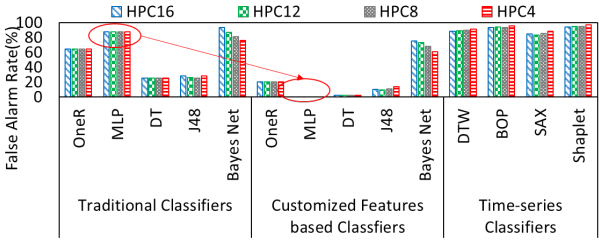

Fig. 6. False alarm rate comparison

IV. Results Evaluation

In this section, we evaluate the effectiveness of *SCARF* detection approach and compare it in terms of various evaluation metrics including detection accuracy and false alarm rate of proposed customized features based classifiers over traditional and time-series classifiers. Lastly, we evaluate the FAM influence on minimizing false alarm rate and reduction of attack detection rate.

A. ML Classifiers Comparison

In this work, time-series classification techniques are adopted for further comparison. To this aim, four prominent time-series classifiers including Dynamic Time Warping (DTW), Bag-Of-Patterns (BOP), Symbolic Aggregate approXimation (SAX), and Shaplet are deployed that are representing various categories of time series classification techniques including shape-based, structure-based without/with order information, and sub-phase shape-based categories. In order to present the effectiveness of using customized features, ML classifiers are fed with only sum value of HPCs named as traditional classifiers. The implemented ML classification algorithms are OneR, multi-layer perceptron (MLP), DecisionTable (DT), J48, and BayesNet that are covering a diverse range of machine learning techniques.

- *Detection Accuracy:* Figure 5-(a) presents the SCA detection accuracy with a varied number of HPCs for the proposed *SCARF* (customized features based classifiers) and existing works (traditional and time-series classifiers using different techniques) [5], [20]. One can observe that the time-series classifiers achieve a lower accuracy despite utilizing more number of HPC features i.e., the SCA detection accuracy is $< 70\%$ on average. Hence, existing time-series classifiers are not the optimal solution for real-time SCA detection. By comparison, the proposed and traditional classifiers achieve above 80% prediction accuracy despite utilizing less number of HPCs, which makes them formidable candidates to consider for real-time SCA detection. Figure 5-(b) zooms in the comparison between proposed and traditional classifiers. It can be seen that *SCARF* method by using customized features based ML classifiers is able to further boost prediction accuracy, ranging from 2% to 6%.

- *False Alarm Rate:* As discussed, despite high detection accuracy, one of the major challenges associated with detection is the false alarms in which we evaluate the false alarm rate for different techniques below. Figure 6 depicts the false alarm rate with proposed and existing techniques when utilizing a varied number of HPCs for SCA detection. The false alarm rates produced by traditional classifiers based SCA detection is significantly high, 57% on an average across all ML techniques and HPC values. This is due to the fact that traditional methods are biased to "under attack". However, the proposed *SCARF* technique with using customized features

employs more features that aid the ML classifiers to predict "under attack" scenario with higher confidence and accuracy. Taking MLP-based SCA detector as an example, the proposed customized classifier can decrease false alarm rate from 87% (obtained when utilizing traditional classifier) to 4.7%, though the detection accuracies are similar. Furthermore, time-series classifiers have shown above 80% false alarm rate.

B. Evaluation of FAM Technique

In this section, we evaluate the attack detection accuracy and false alarm rate of customized features based classifiers with the usage of proposed FAM. As described in Section III-D, setting DN value is based on interval level positive rate, the number of instance intervals, and acceptable false alarm rate. Thus, the DN is set to 2/4 to ensure that false alarm rate is below 30%, 5%. Furthermore, they are compared with DN=1 which corresponds to no "under attack" delay. It

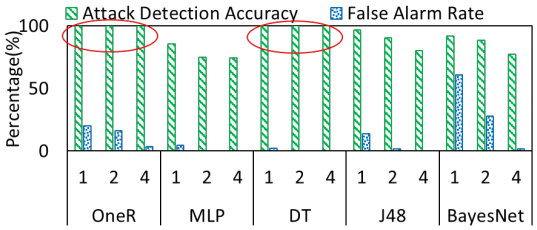

Fig. 7. Attack detection accuracy vs false alarm rate with various DN values

can be seen that increasing the number of DN from 1 to 4 does not have any impact on the attack detection accuracy of OneR and DT classifiers (remains 100%). In the meantime, the false alarm rates of the two classifiers decrease to 3.6% with DN=4 for OneR and 0% with DN=2 for DT classifier. As can be observed for the rest of three ML classifiers, the reduction of false alarm rates can be achieved at the cost of lowering the attack detection accuracy. For instance, in J48 classifier, false alarm rate decreases from 13.6% to 1.7% and to 0%, respectively, while the attack detection accuracy decreases from 96.7% to 90% and then to 80.5% with DN increasing from 1 to 4. It can be concluded that by deploying the proposed FAM technique the false alarm rate can be effectively reduced, while it may cause relative detection accuracy loss for some classification techniques.

V. Conclusion

In this work, we propose *SCARF*, a unified and accurate machine learning-based methodology for detecting SCAs at real-time using the processor's Hardware Performance Counters (HPCs) information. The proposed methodology first solves 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 comprehensive 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. Next, we use HPCs importance evaluation with Correlation Attribute Evaluation algorithm to identify the most prominent HPC features suitable for real-time SCA detection. Furthermore, *SCARF* is further customized with different machine learning classifiers trained specialized set of features to ehnace the accuracy of SCA detection. Lastly, to reduce the false alarm rate, the proposed framework is equipped with False Alarm Minimization (FAM) technique to reduce false alarm rate. Compared to state-of-the-art solutions, *SCARF* shows higher detection accuracy and robustness with lower false alarm rate, achieving 100% attack detection rate with 0% false alarm rate.

References

[1] Brasser, F., and et al. Advances and throwbacks in hardware-assisted security: Special session. In *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems* (2018), IEEE Press, p. 15.

[2] Caruana, R., and Freitag, D. Greedy attribute selection. In *Machine Learning Proceedings 1994*. Elsevier, 1994, pp. 28–36.

[3] Chiappetta, M., Savas, E., and Yilmaz, C. Spectre attack: https://github.com/eugnis/spectre-attack.git.

[4] Chiappetta, M., Savas, E., and Yilmaz, C. Xlate: https://www.vusec.net/projects/xlate/.

[5] 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.

[6] 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.

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

[8] Fulcher, B. D., and Jones, N. S. Highly comparative feature-based time-series classification. *IEEE Transactions on Knowledge and Data Engineering 26*, 12 (2014), 3026–3037.

[9] 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.

[10] Kocher, P., and et.all. Spectre attacks: Exploiting speculative execution. *arXiv preprint arXiv:1801.01203* (2018).

[11] Li, C., and Gaudiot, J.-L. Online detection of spectre attacks using microarchitectural traces from performance counters. In *2018 30th SBAC-PAD*, IEEE.

[12] 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).

[13] 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.

[14] 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.

[15] 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.

[16] 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'20* (2020), IEEE.

[17] Wang, Z., and Lee, R. B. New cache designs for thwarting software cache-based side channel attacks. In *ACM SIGARCH Computer Architecture News* (2007), vol. 35, ACM, pp. 494–505.

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

[19] Yarom, Y., and et.all. Flush+ reload: A high resolution, low noise, l3 cache side-channel attack. In *USENIX Security Symposium* (2014).

[20] Zhang, T., and et.all. Cloudradar: A real-time side-channel attack detection system in clouds. In *RAID* (2016), Springer.

[21] Zhang, T., and Lee, R. B. Secure cache modeling for measuring side-channel leakage. *Technical Report, Princeton University* (2014).

[22] Zhou, B., Gupta, A., Jahanshahi, R., Egele, M., and Joshi, A. Hardware performance counters can detect malware: Myth or fact? In *2018 on AsiaCCS*, ACM.