Run-time Detection of Prime+Probe Side-Channel Attack on AES Encryption Algorithm

Maria Mushtaq
Lab-STICC, University of South Brittany, Lorient, France
maria.mushtaq@univ-ubs.fr

Ayaz Akram
University of California Davis, California, USA
ayaz.akram@wmich.edu

Muhammad Khurram Bhatti
Information Technology University Lahore, Pakistan
khurram.bhatti@itu.edu.pk

Rao Naveed Bin Rais
Ajman University, P.O.Box: 346 Ajman, UAE
r.rais@ajman.ac.ae

Vianney Lapotre
Lab-STICC, University of South Brittany, Lorient, France
vianney.lapotre@univ-ubs.fr

Guy Gogniat
Lab-STICC, University of South Brittany, Lorient, France
guy.gogniat@univ-ubs.fr

***Abstract*— This paper presents a run-time detection mechanism for access-driven cache-based Side-Channel Attacks (CSCAs) on Intel's x86 architecture. We demonstrate the detection capability and effectiveness of proposed mechanism on Prime+Probe attcks. The mechanism comprises of multiple machine learning models, which use real-time data from the HPCs for detection. Experiments are performed with two different implementations of AES cryptosystem while under Prime+Probe attack. We provide results under stringent design constraints such as: realistic system load conditions, real-time detection accuracy, speed, system-wide performance overhead and distribution of error (i.e., false positives and negatives) for the used machine learning models. Our results show detection accuracy of $> 99\%$ for Prime+Probe attack with performance overhead of $3-4\%$ at the highest detection speed, i.e., within $1-2\%$ completion of 4800 AES encryption rounds needed to complete a successful attack.**

***Index Terms*—Side-Channel Attacks, Prime+Probe, Detection, Cryptography, AES, Machine Learning, HPCs.**

I. Introduction

With technologies like Internet-of-Things (IoTs) and Cyber-Physical Systems (CPS) in practice, the age of digital disruption has a renewed challenge of security and privacy. Access-driven Cache-based Side-Channel Attacks (CSCAs) are strong cryptanalysis techniques used to break the otherwise strong cryptographic algorithms by targeting their execution at hardware level [1]. In recent years, Intel's x86 architecture has been exposed to high resolution and stealthy CSCAs such as: Prime+Probe [2], Flush+Reload [3], Flush+Flush [4], Evict & Time [1], Prime & Abort [5], Spectre [6] and Meltdown [7]. Modern-day processors do extensive sharing and de-duplication for performance benefits. CSCAs exploit sharing vulnerabilities in caches [6], [7] to retrieve information. Such attacks rely on the presence of specialized instructions to maneuver the state of shared caches.

The Internet of Things (IoT) rapidly closes the gap between the virtual and the physical world. As significant amount of information is processed through this expanding network, the security of IoT devices and back-end services has become increasingly important. Yet, SCAs pose a significant threat to systems in practice, as the micro-architectures of processors, their access & timing information, power consumption and electromagnetic emanation reveal sensitive information to adversaries. While computational resources and performance are often limited, many IoTs ecosystems deploy back-end servers (often based on Intel's x86 architecture) that collect information for in-depth analysis, advanced customer services, or feedback to actuators in the field. These back-end servers can become a *single-point failure* for entire network if exposed to such SCAs. Recent research shows that entire IoT network can be exposed to SCAs [8] [3] [4] [2] [1] [5].

In order to mitigate against SCAs, both software- and hardware-based protection techniques have been proposed in recent years [9]. Such attacks can be prevented at different levels such as system-level, application-level and hardware-level [9]. Despite many efforts, mitigation techniques against SCAs are still not perfect due to multiple reasons. One of the foremost reasons is the fact that these techniques generally attempt to protect against any given specific vulnerability because an *all-weather* protection against such attacks is often performance costly. Evidence suggest that attacks are becoming sophisticated and stealthier such as Spectre & Meltdown. Therefore, detection techniques can be used as a *first line of defense* against such attacks.

This paper addresses the problem of accurate & early detection of Prime+Probe CSCA at *run-time* using ML techniques in back-end servers that are used for IoT networks. Various linear and non-linear ML models are used to perform detection under stringent design constraints such as: real-time, fast and accurate detection under minimal performance overhead. These ML models use data from HPCs, in near real-time, representing the pattern of memory accesses generated by data-dependent cryptographic operations of AES being carried out by underlying hardware. Using HPCs' data, selected ML models detect Prime+Probe attack during the course of AES encryption rounds. We perform experiments with Prime+Probe attack applied on two different implementations of AES, which is considerably a fast encryption algorithm. The first version is

implemented by Gruss et al. [4], which is slower and recovers half key. The second implementation is a faster version with full key recovery that we have implemented ourselves in order to conduct experiments for this paper. We demonstrate the effectiveness of proposed detection mechanism on Intel's core i7-4770 CPU running on Linux Ubuntu 16.04.1 at 3.40-GHz. We use PAPI (Performance Application Programming Interface) library [10] to access HPCs on Intel's Core $i7$ machines, which offers $100+$ machine-specific events. Since our detection mechanism targets access-driven CSCAs, we considered only those relevant hardware events which are more likely to be affected by Prime+Probe attack. We performed experiments with a set of 12 most relevant events, presented in Section II-C, in order to observe the impact of target computational loads, i.e., crypto-operations and attacks. We also provide guidelines on how only a subset of these 12 counters would be sufficient to train ML models for detection. The main contributions of this paper are as follows:

1) We propose a run-time detection mechanism for Prime+Probe class of access-driven CSCAs. The mechanism uses ML models and execution data of processes at real-time using the HPCs. The novelty in our work comes from the coupling of HPCs with ML models under stringent design constraints such as: real-time, fast and accurate detection with minimal performance overhead.
2) AES encryption algorithm is often used in IoTs. We demonstrate successful detection of Prime+Probe attack on 02 different implementations of AES using Intel's x86 architecture.
3) We provide results under *realistic* system load conditions, i.e., under No, Average and Full Load conditions. These load conditions are achieved by running SPEC benchmarks, which improves the applicability of proposed mechanism.
4) we provide discussion, complimented with experimental results, about detection accuracy, system wide performance degradation, detection speed and distribution of error in terms of false positives and false negatives for selected ML models.

Rest of the paper is organized as follows. Section II presents necessary background & related work. Section III presents the working principle of run-time detection mechanism. Section IV provides experimental evaluation and discussion. Section V concludes this paper.

II. Background and Related Work

A. Cache-based SCAs & Mitigation Techniques

Side channel attacks can destroy robust and substantial cryptosystems such as AES, RSA, ElGamal, DSA and ECDSA by exploiting micro-architectural features of underlying platform. Memory access pattern and timing variations generated by these cryptosystems on the underlying hardware platform are significant sources of information leakage. In the last decade, many cache-based side channel attacks have been proposed that have exposed the micro-architectural vulnerabilities. In the follow-up, many mitigation techniques are also proposed against these attacks [9]. Mitigation solutions being proposed against these attacks are applicable at all levels. For instance, Logical/Physical isolation includes Cache Coloring, Cloudradar, StealthMEM, CacheBar and Hardware Partitioning; Noise-based mitigation includes Fuzzy Times, Bystander Workloads and Anti-correlated Noise; Scheduler-based mitigation includes Obfuscation, Minimum Timeslice and Cache Flushing; Partitioning time mitigation includes Server Side Defences and Kernel Space Isolation; and constant-time techniques include Cache Auditing and Flow-tracker. These strategies, however, often provide protection against specific leakage channel as they introduce significant performance degradation because of cache reservation and blow-up code size. Since, applying mitigation techniques in all cases is expensive, therefore, SCA detection prior to mitigation can help in applying countermeasure on need-basis.

While designing a SCA detection mechanism, the designer should be mindful of three potential problems; 1) Detection process approximates the overall system behavior and it can lead to greater number of false positives and false negatives at run-time, 2) Detection process can slow down the overall program execution and can lead to significant performance overhead at the cost of accurate detection and 3) Detection may be accurate but the detection speed can sometimes be very low, which detects the attack after a theoretical bound of 50% of the attacker activity (which is considered sufficient for a successful attack). We considered all these parameters as evaluation metrics of our run-time detection mechanism.

B. Cache-based SCA Detection

In recent years, some CSCA detection solutions have appeared. Some of them are implemented as user-level processes while others operate at kernel-level. In addition, most of them do not include evaluation of the proposed detection mechanism under realistic system load conditions. Chiappetta et al. in [11] used Neural Networks to build models using HPCs data on benign and spy processes. The proposed mechanism works for Flush+Reload attacks with high accuracy. However, Neural Networks incur heavy performance cost and they are not suitable for run-time early-stage detection. Mathias Payer [12] proposed *HexPADS* which also uses HPCs' data alongwith kernel information like page faults. If the number of cache misses crosses a certain threshold, an attack situation is reported. Experiments using variants of flush+reload and prime+probe attacks show high detection accuracy with low performance overhead for HexPADS. Demme et al. [13] proposed a malware detector that uses used KNN, Decision Trees, Random Forest and Artificial Neural Networks models for prime-probe detection. Authors claim to have no false +ves.

Zhang et al. [14] proposed CloudRadar to detect SCAs in cloud systems by correlating cryptosystem's execution on a VM with the anomalous behavior of caches. Using Prime+Probe & Flush+Reload as case studies, authors claim to have 100% accuracy with the worst case performance overhead less than 5%. Alam et al. proposed a correlation-based detection method for cache and branch predictor based SCAs [15]. Similarly, Bazm et al. [16] relied on Intel cache monitoring technology (CMT) and HPCs while using Gaussian anomaly detection for detection of CSCAs in VMs. The proposed mechanism shows very good accuracy in isolated conditions but suffers from high false positives in noisy conditions. Allaf et al. [17] used KNN model to detect malicious

loop activity within Flush+Reload attack. The proposed model shows achieve good accuracy to detect Flush+Reload attack but it suffers from two issues: firstly, it is performance-costly at run-time due to the working principle of KNN, and secondly, it would not work for other attacks like Prime+Probe. Peng et al. [18] used cache miss rates and data-TLB miss rates to recognize CSCAs. They showed that CSCAs like Flush+Reload have high cache miss rates but low d-TLB miss rates.

C. Selection of HPCs

The HPCs are particular hardware registers, which help to access per-core, per-CPU and system wide profiling for executing processes. Since our interest is to detect access-driven CSCAs, we only consider the events that are plausibly affected by these attacks. In order to best select these events, we performed experiments on a set of 12 best suited events. The set of events are presented in Table I. We collected a system-wide profile for different hardware events under Prime+Probe attack to form a data set of benign and malicious behavior on the system. Data from these counters is used as *features* for ML models. Out of these 12 events, only a subset of $4-5$ events prove to be sufficient as features for our ML models. Thus, we can discard the redundant features and save run-time overheads. Sections III & IV elaborate what kind of information these features offer.

TABLE I
SELECTED EVENTS RELATED TO CSCAS

Scope	Hardware Event as Feature	Feature ID
L1 Caches	Data Cache Misses	L1-DCM
	Instruction Cache Misses	L1-ICM
	Total Cache Misses	L1-TCM
L2 Caches	Instruction Cache Accesses	L2-ICA
	Instruction Cache Misses	L2-ICM
	Total Cache Accesses	L2-TCA
	Total Cache Misses	L2-TCM
L3-Caches	Instruction Cache Accesses	L3-ICA
	Total Cache Accesses	L3-TCA
	Total Cache Misses	L3-TCM
System-wide	Total CPU Cycles	TOT_CYC
	Branch Miss-Predictions	BR_MSP

III. RUN-TIME DETECTION OF PRIME+PROBE

This section provides details on the working principle of detection method being used. Recently, various ML techniques are being used in information security domain such as in [11], [12], [19], [20], [15], [17]. Data classification is an important application of Machine Learning and CSCA detection is basically one such binary classification problem from machine learning prospective. Most of the existing machine learning classifiers can be divided into two basic categories of linear and non-linear models. We experimented with a set of 12 popular classification machine learning models, 6 each from both categories. Table II lists ML models being used in this work.

We performed experiments with all ML models listed in Table II. However, we have finalized only 4 models for our detection mechanism, namely; LDA, QDA, LR and SVM, based on the following reasoning. Although, it is the most important one, classification accuracy is not the only parameter to consider while deploying a high-speed run-time CSCA detection mechanism. We reason that the other most important parameter to examine while comparing ML models is their implementation feasibility. ML models should be easy to embed in the cryptosystem for run-time protection. They should also be able to quickly provide their decision on detection based on the profiling of processes using HPCs. Moreover, the performance overhead caused to the cryptosystem's own execution due to embedding of detection should be reasonable. Upon analyzing the ML models in Table II on these parameters, we have found these 4 models to be most suited ones for run-time, fast & accurate detection with minimal overhead.

We collect the features using HPCs and train these ML models on them. The data provides real-time behavioral information of concurrent processes, including AES as victim, running on Intel's x86 architecture. We select these features based on two factors: 1) Their relevance to the type of SCAs the detection mechanism is targeting and 2) their potential to provide better classification. Based on these criteria, we use L1 data cache misses, L3 total cache misses, L3 total cache accesses and Total CPU cycles as the most relevant features in Prime+Probe detection. These features in our experiments show linearity when the AES encryption operation takes place. For ML models, the LDA is used for reduction in data dimensionality and pattern classification. The focus of this model is to operate a data set into lower dimensional space while providing a distinguishable class for avoiding over-fitting, which is a common problem to data dimensionality. QDA is a non-linear extension of LDA, which builds a quadratic decision surface. Both LDA and QDA assume normal distribution of the measurements. Logistic Regression (LR) is used for defining linear set of relations. The LR model can be used efficiently for prognosis, anticipation and error reduction. It provides a sophisticated binary representation of dependent variables. Whereas, Support vector machine (SVM), a supervised learning model, provides a binary linear classification mapped in such a way that different categories of data are separated or classified with a clear distinction. Selection of specific ML models is based on certain features like; achieving highest accuracy, less computational complexity and less performance overhead during execution under different cryptosystems. We experimented with 12 ML models & selected the best 4 considering the mentioned features.

TABLE II
LIST OF ML MODELS USED FOR CSCA DETECTION

No.	Machine Learning Model	Category
1	Linear Regression (LR)	Linear
2	Linear Discriminant Analysis (LDA)	Linear
3	Support Vector Machine (SVM)	Linear
4	Quadratic Discriminant Analysis (QDA)	Non-linear
5	Random Forest (RF)	Non-linear
6	K-Nearest Neighbors (KNN)	Non-linear
7	Nearest Centroid	Linear
8	Naive Bayes	Linear
9	Perceptron	Linear
10	Decision Tree	Non-linear
11	Dummy	Non-linear
12	Neural Networks	Non-linear

There are three significant phases of our detection mechanism, namely; the Training phase, Run-time profiling and Classification phase and detection phase. In the first phase, we train ML models with HPCs' data collected from two scenarios: one with AES cryptosystem under Prime+Probe attack and the other with no attack. Both scenarios have variable load conditions. For this phase, we use roughly 1-Million data samples with profiling through events excluding kernel events. We configure events within the victim's library to include calls for starting and stopping the HPCs at run-time. In real-time data profiling phase, the sampling granularity has a major influence on victim's (AES) performance when compared to normal execution. Sampling granularity also affects detection speed and accuracy, which are crucial for real-time detection. Our mechanism offers fine- and coarse-grain profiling modes, which provide a trade-off between detection speed and impact on performance. Fine-grain mode collects samples at a very high frequency and it is able to detect Prime+Probe attack relatively early with increased performance overhead. Whereas, coarse-grain mode takes samples at a low frequency, which takes relatively longer to detect the attack but it offers minimal performance overhead. In both cases, our detection module can detect Prime+Probe before its completion. In the detection phase, data from second phase

are passed on to the trained ML models in real-time. Based on these data, each model classifies samples into *Attack* or *No Attack* classes to report intrusion detection. Detection accuracy of each classifier varies depending on the training. Detection speed and performance impact, however, is subject to the sampling frequency.

IV. Experiments and Discussion

We have experimented with two different implementations of Prime + Probe (Prime+Probe) attack on AES cryptosystem. In experiments, we refer them as *PP_Impl1* (slow half key retrieval) and *PP_Impl2* (fast full key retrieval).

A. Detecting Prime+Probe: PP_Impl1

1) Detection Accuracy: Detection accuracy is the most important indicator to assess a CSCA detection mechanism. We use unbiased training data with equal number of attack and no-attack samples. Table III shows the detection accuracy of the selected ML models. The detection accuracy is very high for all ML models (close to 100%) under all load conditions. The only exception is LDA under NL and AL, where it still shows a detection accuracy above 95%. In order to explain this high accuracy of all ML models we can have a look at Figure 1 and 2, which show the distribution of hardware events. In these figures, Y-axis depicts frequency of samples and X-axis depicts magnitude of measured events. The results in green color present normal execution of cryptosystem under no attack. Whereas, the results in red color present abnormal execution of cryptosystems under attack scenario. As visible in Figure 1, all used features show clearly distinctive behavior under NL resulting into easy classification for ML models. Under FL condition (shown in Figure 2), the used hardware events start to overlap. However, two features (L3's total cache accesses total cache misses) still exhibit distinctive behavior leading to good performance of ML models.

TABLE III
Results using selected models for Prime+Probe (PP_Impl1)

Model	System Condition	Accuracy (%)	Speed (%)	FP (%)	FN (%)	Overhead (%)
LDA	NL	95.15	2.1	0	4.85	3.48
	AL	97.47	2.1	0	2.53	
	FL	100	1.1	0	0	
LR	NL	99.89	2.1	0.11	0	3.23
	AL	99.97	2.1	0.03	0	
	FL	99.92	2.1	0.08	0	
SVM	NL	100	2.1	0	0	5.08
	AL	100	2.1	0	0	
	FL	99.99	2.1	0	0.01	
QDA	NL	100	1.1	0	0	1.68
	AL	99.99	1.1	0.01	0	
	FL	99.99	2.1	0.01	0	

2) Detection Speed: Detection speed usually depends on the sampling resolution of detection mechanism. This resolution also impacts the performance overhead. In order to reliably estimate upper 4-bits of a secret key byte, Prime+Probe attack needs at least 4800 AES encryption rounds [4]. Therefore, the detection of Prime+Probe would be useful only if it is achieved before completion of 4800 encryption rounds. Here, we define the detection speed as number of encryption rounds needed to detect the attack, taken as a percentage of 4800 encryption rounds (i.e., the upper bound). For instance, a detection speed of 2.1% would mean that detection is achieved within first 100 encryption rounds. Table III shows the run-time detection speed achieved by all ML models while detecting PP_Impl1. Our ML models are able to detect the attack within first 100 encryption rounds, which is well ahead of 4800 AES encryption rounds under all load conditions.

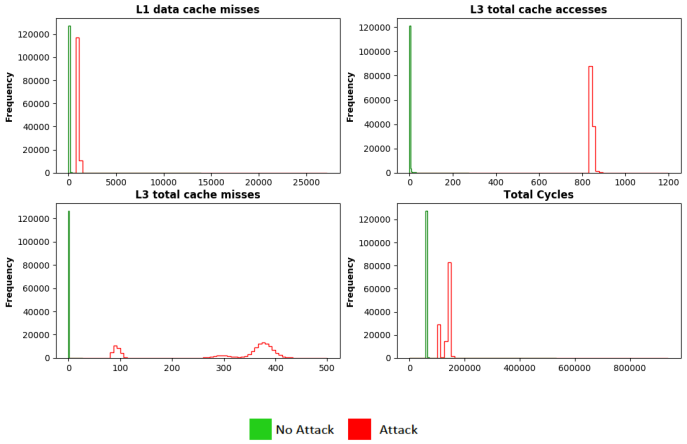

Fig. 1. Selected HPCs under NL condition for AES encryption: With & Without Prime+Probe Attack (*PP_Impl1*)

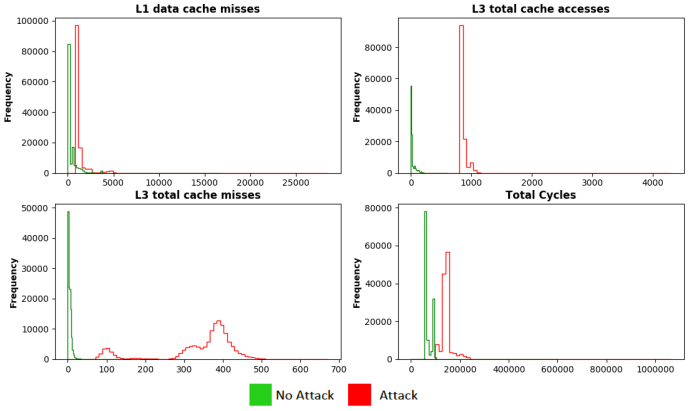

Fig. 2. Selected HPCs under FL condition for AES encryption: With & Without Prime+Probe Attack (*PP_Impl1*)

3) Confusion Matrix: The CSCA detection is a classification problem. Therefore, detection inaccuracy can be split into false positives & negatives to analyze detection results. Table III shows the detection inaccuracy of our ML models under all load conditions. The percentage of false positives and negatives out of the evaluated samples is very close to 0 in almost all cases.

4) Performance Overhead: Performance overhead, in terms of slowdown, is a pertinent aspect to look into while embedding the CSCA detection mechanism within encryption module. It determines the applicability of the proposed detection mechanism. Performance overhead is linked with the detection speed as higher resolution can lead to higher performance overhead. Table III shows that the performance overhead is generally low for our ML models while performing run-time detection. We sample hardware events every 50 encryption rounds to make detection decisions. Since, the detection speed is already very high, the sampling frequency of performance counters can be relaxed which would lead to further reduction in performance overhead.

B. Detecting Prime+Probe: PP_Impl2

1) Detection Accuracy: Table IV shows the accuracy of used ML models while detecting second implementation of Prime+Probe attack on AES. Under all load conditions the ML models are able to show pretty high accuracy (above 99%). Figure 3 and 4 show the distribution of HPCs used for detection under attack and no-attack cases for NL and FL system conditions respectively. As indicated in these figures even under high load condition the distinction among the used HPCs is good enough for detection of attack.

2) Detection Speed: Table IV illustrates that all ML models show very high detection speed for the faster implementation of Prime+Probe attack. The attack is detected at maximum by 100

encryption rounds in all cases, proving the capability of a prompt response by our detection mechanism.

TABLE IV
RESULTS USING SELECTED MODELS FOR PRIME+PROBE (*PP_Impl2*)

Model	System Condition	Accuracy (%)	Speed (%)	FP (%)	FN (%)	Overhead (%)
LDA	NL	99.96	2.1	0.003	0.037	3.59
	AL	99.83	1.1	0.12	0.051	
	FL	98.03	2.1	1.86	0.11	
LR	NL	99.95	2.1	0.015	0.034	4.02
	AL	99.70	2.1	0.28	0.020	
	FL	94.82	2.1	5.12	0.061	
SVM	NL	99.97	2.1	.0037	0.026	3.85
	AL	99.96	2.1	0.012	0.028	
	FL	99.94	25.1	0.035	0.025	
QDA	NL	100	2.1	0	0	3.80
	AL	99.99	1.1	0.01	0	
	FL	99.99	1.1	0.01	0	

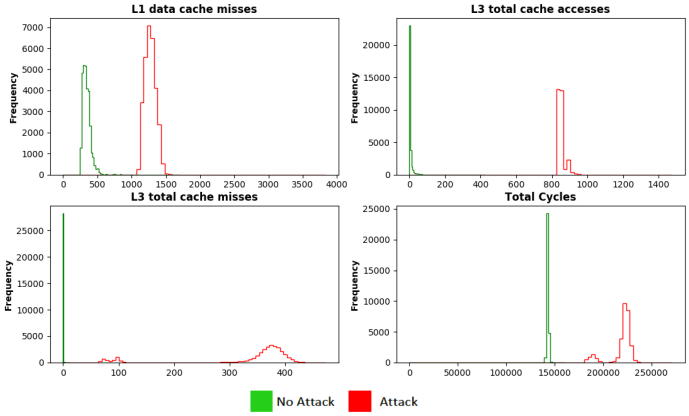

Fig. 3. Selected HPCs under NL condition for AES encryption: With & Without Prime+Probe Attack (*PP_Impl2*)

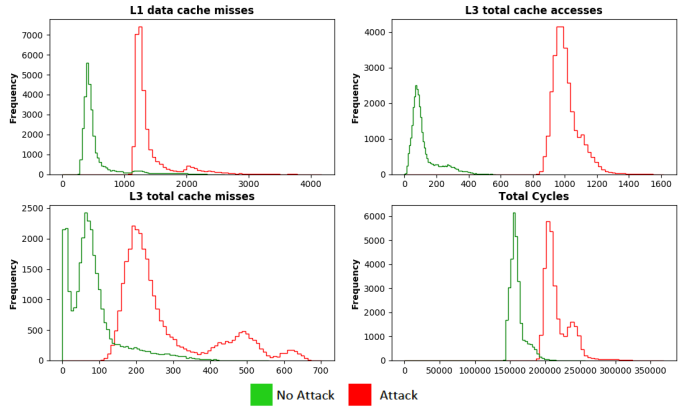

Fig. 4. Selected HPCs under FL condition for AES encryption: With & Without Prime+Probe Attack (*PP_Impl2*)

3) Confusion Matrix: Table IV shows the division of wrongly classified samples by ML models into false positives & negatives. In the majority cases, the %ages of false +ves & -ves is very low i.e., the inaccuracy is very low. In a couple of cases (LR & LDA with FL) where inaccuracy is considerable, the majority of the mis-classified samples belong to false positives.

4) Performance Overhead: Finally, the performance overhead (shown in Table IV) of all ML models to perform online detection of *PP_Impl2* attack is low and insignificant. Moreover, this detection overhead can be further reduced by reducing the sampling frequency of performance counters used for detection of the attack.

V. CONCLUSION

This paper presents experimental evaluation and the results on the run-time detection of Prime+Probe CSCA on Intel's x86 architecture using multiple machine learning models that collect real-time data from the HPCs for fast & accurate detection on AES cryptosystem. We argue that intelligent performance monitoring of concurrently executing processes at hardware-level, coupled with machine learning methods, can enable early detection of high precision and stealthier CSCAs. Our evaluation metric comprises of detection accuracy, speed, system-wide performance overhead, realistic system load conditions and confusion matrix for ML models. Our results show detection accuracy of $> 99\%$ for Prime+Probe attack with performance overhead of $3-4\%$ at the highest detection speed, i.e., within $1-2\%$ completion of 4800 AES encryption rounds needed to complete the attack.

REFERENCES

[1] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: The case of aes," *CT-RSA*, pp. 1–20, 2006.

[2] M. S. Inci, B. Gulmezoglu, G. Irazoqui, T. Eisenbarth, and B. Sunar, "Cache attacks enable bulk key recovery on the cloud," vol. 9813. Santa Barbara, CA, USA: CHES, 08 2016, pp. 368–388.

[3] Y. Yarom and K. Falkner, "Flush+reload: A high resolution, low noise, l3 cache side-channel attack," in *USENIX Security 14*, p. 719.

[4] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+flush: A fast and stealthy cache attack," in *DIMVA*, 2016, pp. 279–299.

[5] C. Disselkoen, D. Kohlbrenner, L. Porter, and D. Tullsen, "Prime+abort: A timer-free high-precision l3 cache attack using intel TSX," in *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, 2017, pp. 51–67.

[6] P. K. et al, "Spectre attacks: Exploiting speculative execution," 2018.

[7] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown."

[8] "https://threatpost.com/mixed-signal-microcontrollers-open-to-side-channel-attacks/134793/," 2018.

[9] Q. Ge, Y. Yarom, D. Cock, and G. Heiser, "A survey of microarchitectural timing attacks and countermeasures on contemporary hardware," *IACR Crypt. ePrint Arch.*, p. 613, 2016.

[10] "Papi," 2018, http://icl.cs.utk.edu/papi/.

[11] M. Chiappetta, E. Savas, and C. Yilmaz, "Real time detection of cache-based side-channel attacks using hardware performance counters," *Appl. Soft Comput.*, vol. 49, no. C, pp. 1162–1174, Dec. 2016.

[12] M. Payer, "Hexpads: a platform to detect stealth attacks," in *International Symposium on Engineering Secure Software and Systems*. Springer, 2016, pp. 138–154.

[13] J. Demme, M. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan, and S. Stolfo, "On the feasibility of online malware detection with performance counters," in *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3. ACM, 2013, pp. 559–570.

[14] T. Zhang, Y. Zhang, and R. B. Lee, "Cloudradar: A real-time side-channel attack detection system in clouds," in *International Symposium on Research in Attacks, Intrusions, and Defenses*, 2016, pp. 118–140.

[15] M. A. et al, "Performance counters to rescue: A machine learning based safeguard against micro-architectural side-channel-attacks," Crypt. ePrint Arch., 2017, https://eprint.iacr.org/2017/564.

[16] M.-M. Bazm, T. Sautereau, M. Lacoste, M. Sudholt, and J.-M. Menaud, "Cache-based side-channel attacks detection through intel cache monitoring technology and hardware performance counters," in *IEEE FMEC, 3rd Int'l Conf. on*.

[17] Z. Allaf, M. Adda, and A. Gegov, "Confmvm: A hardware-assisted model to confine malicious vms," in *UKSim2018: UKSim-AMSS 20th International Conference on Modelling & Simulation*. IEEE, 2018.

[18] S.-h. PENG, Q.-f. ZHOU, and J.-l. ZHAO, "Detection of cache-based side channel attack based on performance counters," *DEStech Transactions on Computer Science and Engineering*, no. aiie, 2017.

[19] M. Mushtaq, A. Akram, M. K. Bhatti, M. Chaudhry, V. Lapotre, and G. Gogniat, "Nights-watch: A cache-based side-channel intrusion detector using hardware performance counters," in *ISCA'18 Int'l Workshop on Hardware and Architectural Support for Security and Privacy*. Los Angeles, CA, USA: ACM, 2018, pp. 1:1–1:8.

[20] Z. Allaf, M. Adda, and A. Gegov, "A comparison study on flush+reload and prime+probe attacks on aes using machine learning approachess," *UK Workshop on Computational Intelligence*, pp. 203–213, 2017.