# Sherlock Holmes of Cache Side-Channel Attacks in Intel's x86 Architecture

Maria Mushtaq, Ayaz Akram, Muhammad Khurram Bhatti, Usman Ali, Vianney Lapotre, Guy Gogniat

# Sherlock Holmes of Cache Side-Channel Attacks in Intel's x86 Architecture


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

Ayaz Akram
*University of California*
Davis, USA
yazakram@ucdavis.edu

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

Usman Ali
*Information Technology University*
Lahore, Pakistan
msee17009@itu.edu.pk

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

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



***Abstract*— Intel's x86 architecture has been exposed to high resolution and stealthy cache side channel attacks (CSCAs) over past few years. In this paper, we present a novel technique to detect CSCAs on Intel's x86 architecture. The proposed technique comprises of multiple machine learning models that use real-time behavioral data of concurrent processes collected through Hardware Performance Counters (HPCs). In this work, we demonstrate that machine learning models, when coupled with intelligent performance monitoring of concurrent processes at hardware-level, can be used in security for early-stage detection of high precision and stealthier CSCAs. We provide extensive experiments with four variants of the state-of-the-art CSCAs. We demonstrate that our proposed technique is resilient to noise generated by the system under various loads. To do so, we provide results under realistic system load conditions with an evaluation metric comprising of detection accuracy, speed, system-wide performance overhead and confusion matrix for machine learning models. In experiments, our technique achieves detection accuracy of up to 99.51% for Flush+Reload attack on RSA, incurring a performance overhead of 1.63% and 99.99% accuracy on AES while incurring a maximum performance overhead of 8.28%. Our experimental results show consistency for Flush+Flush attack on different implementations of AES as well.**



***Index Terms*—Cryptography, Side-Channel Attacks, Detection, Machine Learning, RSA, AES, Flush+Reload, Flush+Flush.**


## I. Introduction

Side-Channel Attacks (SCAs) target cryptographic implementations for unauthorized retrieval of information [1], [2], [3]. The threat of side channel leakage imposes a serious concern to data privacy as it can break the otherwise theoretically sound cryptographic algorithms at implementation-level. For performance benefits, extensive sharing and de-duplication is performed in modern processors. The CSCAs is a sub-class of SCAs, which exploits sharing vulnerabilities in caching hardware to retrieve confidential information. These attacks exploit specialized instructions to manipulate the state of shared caches such as LLC. Numerous mitigation techniques have been proposed against various threats generated by SCAs [4], [5] both at software and hardware levels. These techniques suggest to prevent such attacks at various levels in computing hierarchy [4]. For instance, at the system level, physical and logical isolation approaches exist [6], [7]. Whereas, at the hardware level, it is rather difficult to apply mitigation techniques due to the fact that they are not favorable on commodity systems in terms of cost and complexity of their design. Nevertheless, hardware solutions are mostly based on having new secure caches, changes in prefetching policies and either randomization or complete removal of cache interference [8], which would cause drastic changes the way computing systems work. At the application level, mitigation solutions mainly target the source of information leakage [9] and try to obfuscate by either randomizing or making side-channel information constant (timing, power, access etc.). Despite many efforts, the proposed mitigation techniques against SCAs are not effective. This is mainly because these techniques generally offer protection against some specific vulnerability and do not take a system-wide approach. Moreover, a one-for-all solution against such vulnerabilities is often costly in terms of performance. The proposition of recent attacks like Spectre [10] and Meltdown [11] are evidences that side-channel attacks are becoming even more sophisticated and stealthier [2] over the time. Detection techniques, on the other hand, can help applying mitigation only when needed and thus, can save the hard-earned performance benefits of existing computing systems. However, for detection-based prevention strategy to be effective, detection needs to be highly accurate, should incur minimum system overhead at run-time, should cover a large set of attacks and should be capable of early-stage detection, i.e., before the attack completes.

In this paper, we address the problem of accurate & early detection of CSCAs at *run-time*. We present a novel detection mechanism that uses machine learning models. These models use PHC data in near real-time using selected hardware event. The data represent memory access pattern generated by data-dependent cryptographic operations that are being carried out by underlying hardware. Using data from hardware events as features, these ML models detect CSCAs concurrently during

the course of encryption. We perform experiments with three state-of-the-art access-driven CSCAs, Flush+Reload on RSA [1] & AES [12] and Flush+Flush [2] on AES cryptosystem with two different implementations (slow half key recovery and fast implementation of full key recovery). The effectiveness of proposed detection mechanism has been demonstrated on Linux Ubuntu 16.04.1 running on Intel's core i7-4770 CPU at 3.40-GHz in our expeirments. Following are the the main contributions of this paper:

1) We propose a generic run-time detection technique for CSCAs that uses multiple machine learning models. It relies on run-time profiling of concurrent processes, which are collected directly through the hardware events using HPCs in near real-time. The HPCs are used in previous research work as well. However, the novelty in this work comes from the selection and coupling of suitable hardware events with machine learning models under stringent constraints such as: run-time, fast and accurate detection with minimal performance overhead under realistic system load conditions and against a larger set of stealthier CSCAs.
2) We demonstrate successful detection of three variants of state-of-the-art CSCA implementations, i.e., Flush+Reload on RSA, Flush+Reload on AES and Flush+Flush on AES cryptosystems.
3) We demonstrate that the proposed technique is resilient to noise generated by the system under various loads. To do so, the results are provided under *variable* system load conditions, i.e., under No Load (NL), Average Load (AL) and Full Load (FL) conditions. In order to achieve such load conditions, memory-intensive *SPEC* benchmarks are being run on the system along with the cryptosystem and attacks.
4) We provide guidelines on selecting appropriate hardware events and machine learning models for run-time detection of CSCAs.

Rest of the paper is organized as follows. Section II presents related work on CSCAs, their mitigation and detection techniques. Section III provides necessary background. Section IV presents the proposed technique including the selection of machine learning models and methodology. Section V provides experimental evaluation and discussion. Section VI discusses comparative analysis of our detection mechanism with state-of-the art. Section VII concludes this paper.

## II. Related Work

### A. State-Of-The-Art on CSCAs and Mitigation Techniques

The CSCAs exploit micro-architectural features of hardware platforms to attack cryptosystems such as RSA, AES, DSA, ElGamal, ECDSA etc. Significant source of information extraction in such attacks is due to the recognition of memory access pattern and timing variation. In the last decade, many CSCAs have been seen and in the follow-up, many mitigations were proposed as a solution to these attacks [4], [5]. Some examples of attack categories are Flush+Reload [1], Flush+Flush [2], Prime & Probe [13], Evict & Time [3] and Evict & Reload [14]. In the recent past, solutions to mitigate SCAs have been proposed from time to time such as: Logical/Physical isolation (including Cache Coloring [15], CloudRadar [16], STEALTH-MEM [9], cacheBar [17], hardware partitioning [18]), Noise based mitigations [19], Scheduler-based countermeasures [20], Partitioning time [21], [22]. These mitigation techniques often provide protection against specific leakage channels. Moreover, they drastically increase performance overhead and code size while working under attack scenario. As a performance versus security trade-off, applying countermeasure as an *all weather* approach is often expensive because they require entirely new system design in order to maintain performance benefits. Thus, detection methods can facilitate in indicating the use of countermeasure on need-basis.

### B. State-Of-The-Art on Detection Mechanisms

There are a few research works on detecting CSCAs. Some of these solutions are implemented as user-level processes while others operate at kernel-level. Most of these works do not include detection on latest CSCAs like Flush+Flush. One of the fundamental limitation of proposed detection mechanisms is that they do not include evaluation under noisy backgrounds, i.e., under more realistic system load conditions. Some of the mechanisms are threshold based which are not very sophisticated for decision making. Many detection mechanisms talk about their accuracy but they lack the discussion of their mechanism in terms of detection speed, percentage of miss-classification, effect on realistic load conditions and performance overhead. In contrast, we evaluate our proposed detection mechanism on a wide set of high resolution and stealth attacks including two variants of Flush+Flush. We also use varying system load conditions to show robustness of the proposed detection mechanism.

Chiappetta et al [23] proposed to use HPCs in conjunction with Neural Networks for building models for benign and spy processes. They have shown that the proposed mechanism works well for Flush+Reload attack with high accuracy. Mathias et al. [24] proposed a detection solution named *HexPADS*, which uses data from different HPCs along-with kernel information like page faults. Their experiments with Prime+Probe and Flush+Reload attacks show 100% accuracy and less than 2% performance overhead. However, paper lacks the discussion on detection speed that how fast *HexPADS* was able to detect attack. Whereas, *HexPADS* assumes no realistic load on the system, therefore, overhead may increase with realistic scenarios. Threshold determination is not a very sophisticated way for detection because threshold determination can vary with different load conditions.

Zhang et al [16] proposed a framework, called *CloudRadar*, which is implemented as a patch to the OS. Although, CloudRadar has demonstrated a 100% accuracy with the performance overhead of less than 5% for Prime+Probe and Flush+Reload attacks, it detects these attack in isolated conditions. That is, no tests under realistic/noisy scenario have been performed to see the efficiency of detection mechanism.

Allaf et al [25] detects Flush+Reload and Prime+Probe attack at 97% and 98% detection accuracy at speed of 2% under SPEC benchmarks but proposed system does not identify the impact on performance of the proposed mechanism.

In the work proposed by Bazm et al. [26], they use HPCs and Intel Cache Monitoring Technology (CMT). They have used gaussian anomaly detection for detection of Prime+Probe in VMs. Their solution demonstrates good accuracy under no load consitiond but suffers from high rate of false positives under noisy conditions. The work of Peng et al. [27] ueses Cache miss rate and data-TLB miss rate to detect CSCAs. This work fundamentally argues that CSCAs exhibit high cache miss rate and low data-TLB miss rate. Another recent tool, named as *SCADET* [28] is a signature-based detection mechanism which takes Prime+Probe as a case study. Instead of using HPCs, this approach uses high-level semantics and invariant patterns of attack. Results reveal that *SCADET* provides high 100% accuracy but it lacks discussion on detection speed and performance overhead of the mechanism. Authors report that, in some cases, system provides false alarms under load conditions. Moreover, the trace analysis time of this approach for detection is very long (notable irregularities when trace exceeds a certain size), under such conditions, using *SCADET* for run-time detection is debatable.

## III. Background Knowledge

This section provides the fundamental background needed to understand the methodology, experiments and results of our proposed solution. At first, in this section, we demonstrate the working and effectiveness of three most recent CSCAs that are used in this paper for experiments and validation of our proposed detection mechanism [1], [12], [2].

### A. Flush+Reload Attack on RSA Cryptosystem

Flush+Reload attack [1] on RSA cryptosystem exploits the vulnerability of *demand-fetch* policy of caches by tracking the difference in timing, in terms of cache *hit* or cache *miss*, when data is loaded into cache. Flush+Reload attack is mainly dependent on two factors: 1) the permissions to *flush* any cache line by virtual address on targeted architecture and 2) the presence of shared libraries that victim and attacker processes both are sharing while executing on Intel's $x86$ architecture (attack has been demonstrated on Intel $x86$) having inclusive caches. Working principle of Flush+Reload attack is elaborated in detail in [1].

### B. Flush+Reload Attack on AES Cryptosystem

Compared to RSA, AES is a considerably fast cryptosystem. Recently, Flush+Reload attack has also been implemented on AES [12], [29]. It uses a fast and light implementation of T-table in OpenSSL as compared to other proposed attack implementations on AES. In this case, Flush+Reload attack mainly focuses on the last round of AES encryption to collect round key, which provides original secret key by reversing the key expansion algorithm of AES. Further details on this attack can be found in [12].

### C. Flush+Flush Attack on AES Cryptosystem

Flush+Flush [2] is a cross-core SCA that works in virtualized environment. We have used two variants of Flush+Flush attack for experiments as well. One implementation with half key retrieval and another implementation with full key retrieval, which happens to be faster. Working principle of Flush+Flush attack is similar to Flush+Reload attack, except the fact that it is significantly stealthier due to flushing operation alone, i.e., with no reload operation. It exploits the same hardware and software vulnerabilities as Flush+Reload attack does, except that it targets OpenSSL T-Table AES implementation. Unlike other CSCAs, Flush+Flush does not perform significant memory accesses, which reduces the number of additional cache misses with minimal cache hits. Authors of Flush+Flush attack [2] tried to detect their own attack using hardware events in Linux but they declared that their attack is non-detectable with at least 24 different hardware events in Linux syscall interface. We tried two implementations of Flush+Flush to make sure the accurate detection of this stealthy and non-detectable attack. In this work, we show that Flush+Flush attack is detectable using our proposed mechanism. We have taken one implementation of Flush+Flush attack from their authors [2], which targets first round of encryption to recover half-key. Whereas, for the second implementation, we have taken Flush+Reload on AES implementation from [12] and modified it ourselves for faster and full-key recovery. Further details on Flush+Flush attack are present in [2] and [12].

## IV. Proposed Run-time Detection Mechanism

The challenge of designing a detection mechanism for CSCAs is three-pronged. Firstly, detection mechanism would slowdown in terms of overall encryption time and thus, it can lead to significant performance overhead while trying to achieve higher detection accuracy. Secondly, an accurate but late detection is useless for run-time detection mechanisms. Theoretically, detection of an attack after 50% of its completion is considered as sufficient for a successful attack [3], [30]. Thus, detection speed is equally important for run-time adaptation of such mechanisms. And thirdly, a detection mechanism must not lead to a higher number of False Positives (FPs) and False Negatives (FNs) at run-time. We considered all these aspects in our evaluations of the proposed run-time detection mechanism.

### A. System Model

We have performed our proposed detection mechanism on Linux Ubuntu 16.04.1 with kernel $4.13.0-37$ running on Intel's core $i7-4770$ CPU at 3.40-GHz. We validate our detection mechanism on access driven CSCAs which are the major threat for information leakage in cache hierarchy of Intel's architecture. In our detection mechanism, threat model is *same-core* and *cross-core* SCAs. It assumed that operating system is not compromised. Information of hardware events can be retrieved by high-level software libraries/APIs such as; PerfMon, OProfile, Perf tool, Intel Vtune Analyzer and PAPI. We have used PAPI (Performance Application Programming Interface) [31] library to access HPCs on Intel Core $i7$ machine.

Section IV-B provides details on selection of hardware events using PAPI libraries for our machine learning models.

### *B. Selected Hardware Events as Useful Features*

Hardware performance counters are supported by almost all modern processors today. In order to access these counters, software APIs are used. Performance API (PAPI) is one such library that provides access to hundreds of events for Intel's architecture. These events provide access to core-wide, CPU-wide and system-wide profiles of concurrent processes. Since the scope of this work is limited to detection of access-driven CSCAs, therefore, we consider only the events that are plausibly affected by these attacks. The set of events that we have tested includes total cache accesses and misses for all levels of caches such as: L1 data cache misses (L1-DCM), L2 total cache accesses (L2-TCA), L3 instruction cache accesses (L3-ICA) etc. and other pipeline events like total execution cycles (TOT-CYC) and branch mispredictions (BR-MSP). Using these events, we collect a system-wide profile for both benign and malicious processes while running state-of-the-art CSCAs on Intel's x86 architecture.

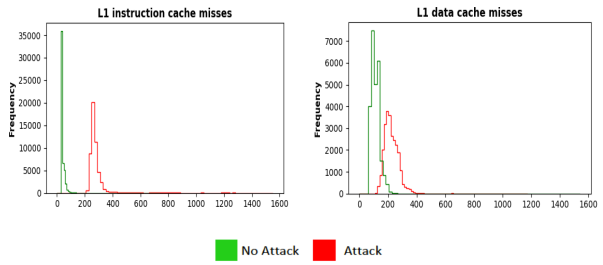


Fig. 1: Experimental results of two hardware events for Flush+Reload (on RSA) and Flush+Flush (on AES) attacks.

Figure 1 illustrates the experimental data collected from two hardware events that capture the behavior of L1 instruction (left) and L1 data (right) cache misses during the operations of RSA and AES cryptosystems while running under attack and no attack scenarios. The attacks used to obtain these results are Flush+Reload on RSA and Flush+Flush on AES cryptosystems. Sampling frequency is shown on Y-axis, while magnitude of measured events is shown on X-axis. Data in green represents the execution of target cryptosystem under no attack, whereas, in red it represents attack scenario.

Since, Flush+Reload on RSA is based on Flush+Reload of instructions in caches, Figure 1 shows that the magnitude of instruction cache misses largely shifts to right under attack creating a clear distinction between attack and no-attack cases. Similarly, in case of Flush+Flush on AES, which is based on flushing of data from caches, the magnitude of data cache misses shifts to right under attack and a segregation between attack and no-attack scenarios becomes visible. Although these two cases (which show events under No-Load) seem easier to handle for a detection mechanism, the distribution of events can become more complicated under varying system loads and that is where Machine Learning can help. The complete list of

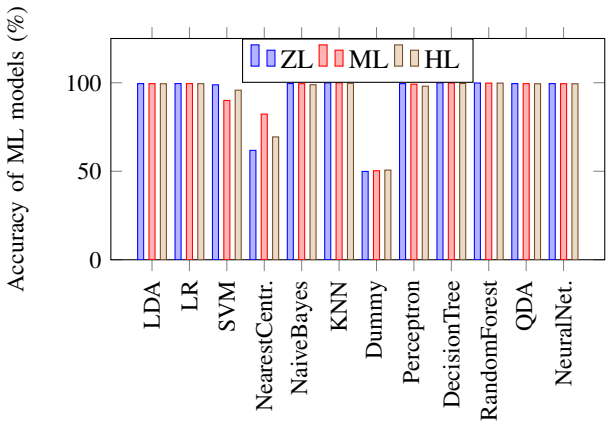


Fig. 2: Accuracy Comparison of machine learning models for Flush+Reload (RSA) detection

hardware events used for detection of attack on RSA include: L1-instruction cache misses, L3-instruction cache accesses, L3-total cache misses and total execution cycles. The list of hardware events for detection of attacks on AES include: L1-instruction cache misses, L1-data cache misses, L3-total cache misses and total execution cycles.

### *C. Selection criteria for machine learning models*

In order to select most suitable models, experiments have been performed 12 different machine learning classifiers from both linear and non-linear categories as shown in Figure 2 and 3. A detailed review of these machine learning models can be found in [32]. Here we discuss in detail, the rationale behind selecting a sub-set of those models for further experimentation. We rely on two basic parameters to choose machine learning models: classification accuracy and implementation feasibility for run-time detection. Figures 2 and 3 show the classification accuracy of all machine learning models while detecting Flush+Reload on RSA and Flush+Flush on AES, respectively. In the first stage, we selected the machine learning models that showed acceptable accuracy for detection of both attacks. This leaves us with Linear Discriminant Analysis (LDA), Logistic Regression (LR), Support Vector Machine (SVM), Naive Bayes, K-Nearest Neighbor (KNN), Decision Tree, Random Forest and Quadratic Discriminant Analysis (QDA). Since LDA and QDA are extensions of Naive Bayes, we let go Naive Bayes and picked LDA and QDA.

Decision Tree and Random Forest shown very good detection accuracy but they are not easy to implement at run-time due to their tree-based nature which results in comparatively higher performance overheads than linear models. Similarly, KNN needed to use all data of the training set while performing inference at run-time which leads to excessive storage and performance overhead. Therefore, we picked LDA, LR, SVM and QDA as final candidates for use in the proposed SCA detection framework. Features collected from hardware events show linearity when the encryption operation takes place under RSA and AES cryptosystems.

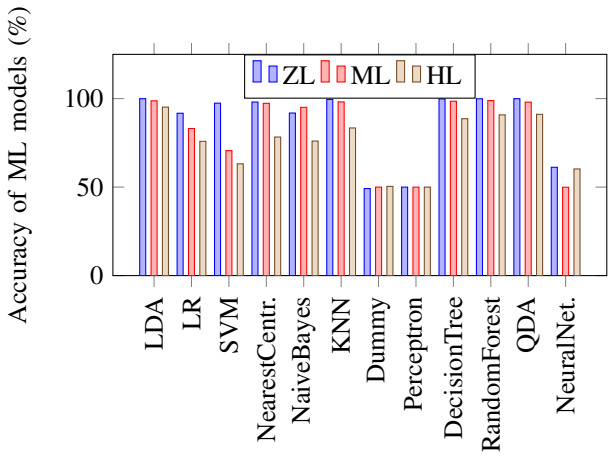


Fig. 3: Accuracy Comparison of machine learning models for Flush+Flush (AES) detection.

### D. Methodology

An abstract representation of our proposed detection mechanism with LDA, LR, SVM and QDA machine learning models is given in Figure 4. There are three significant steps of our detection mechanism, namely, Training of machine learning models, Run-time profiling and Classification & detection. In the following, we explain each step in detail.

*1) Training of machine learning models:* The events which are used for profiling (discussed in previous section), provide us with a differentiation between attacking and non-attacking processes for RSA and AES cryptosystems. We collected training data of 1-Million samples from attack & no attack execution scenarios using variable load conditions, which helped us to train our machine learning models with this classified data. Our training data contains equal number of samples of both attack and no-attack scenarios. Sample size of 1-Million for attack and no-attack samples is sufficient enough to learn the small variations in victim's behavior. We have to apply training process once so that ML models learn the behavior of attack by every possible execution scenario (realistic load conditions). Once the attack behavior is learned, it takes hardware events at run-time and detects attack on the go. For validation purposes, we applied $K-$fold cross validation technique [32] for all the models on the training data to verify them before application on run-time detection.

*2) Run-time Profiling:* In the second phase, our detection mechanism collects run-time samples from the selected hardware events. Our detection mechanism collects run-time samples of hardware events by offering fine-grained and coarse-grained profiling modes. In fact, these two modes of profiling offer a trade-off between performance and the speed of detection. For instance, in fine-grained mode, samples from hardware events are collected at a higher frequency, which subsequently leads to an early-stage detection of attacks but at an increased performance overhead. In coarse-grain profiling mode, the data samples are taken at a low frequency, which takes longer time in detecting attacks. In this mode, however, the performance overhead is minimal as the data samples from hardware events are collected less frequently. Our detection mechanism demonstrates successful detection in both cases, i.e., before the completion of attack.

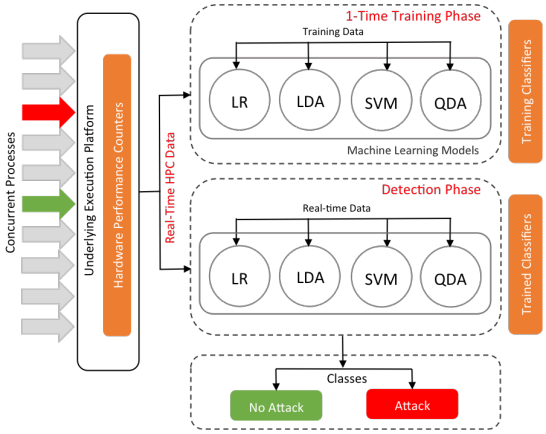


Fig. 4: Abstract view of detection mechanism.

*3) Classification & Detection:* In the third and last phase, our detection mechanism utilizes run-time collected data from previous phases for classification and detection purpose. Collected data from run-time profiling phase is passed to the machine learning models and classifiers at real-time with prefixed frequency which serve as abnormality/anomaly detector. On the basis of this trained data, every model classifies the data into two categories: *Attack* or *No Attack*. Detection accuracy is based on how well the model is learned or trained, whereas, speed of detection and performance is dependent on how fast we try to collect the samples.

## V. Experiments and Discussion

In order to validate our proposed detection mechanism, we demonstrate three case studies with extensive experiments: 1) Detection of Flush+Reload attack on RSA cryptosystem, 2) Flush+Reload attack on AES cryptosytem and 3) Flush+Flush attack on 2 different implementations of AES cryptosystem (one is slow implementation with half key retrieval referred as *FF_Imp1* and the other is a faster version with full key retrieval referred as *FF_Imp2*). All attack implementations on AES have been tested under OpenSSL 0.9.7 and 1.0.1f versions whereas, on RSA, Flush+Reload attack is tested under GnuPG 1.4.13 version as stated in the state-of-the-art. Our detection mechanism efficiently detects Flush+Reload and Flush+Flush attack techniques and will work the same way for its variants too at run-time. In these case studies, we assess the detection capability of machine learning models under realistic load conditions. Current state-of-the-art demonstrates the effectiveness of attacks and their detection & mitigation solutions when victim and attacker are the only running processes and there is no other load on the system. In practice, however, the system is often running other processes concurrently with the encryption operation. Thus, we examine our proposed mechanism under realistic system load. We have generated these variable system loads by concurrently executing selected SPEC benchmarks in the background [33], such as *gobmk*, *mcf*, *omnetpp* and *xalancbmk*, which offer memory-intensive

computations. There are three load conditions, namely; the *Zero Load (ZL)* condition is the same as used in the state-of-the-art that includes victim and attacker processes being the only load on the system, the *Medium Load (ML)* condition includes at least 2 benchmarks processes executing along-with victim and attacker processes and the *Heavy Load (HL)* condition includes at least 4 SPEC benchmarks processes executing along-with victim and attacker processes. We collect hardware events at run-time in fine-grain and coarse-grain detection modes. In fine-grain mode, hardware events are collected after every 10 encryptions whereas, in coarse-grain mode, hardware events are collected after every 50 or 100 encryptions. In this paper, we present the results with fine-grain sampling frequency in order to showcase that our detection mechanism is capable of providing early detection of CSCAs.

### A. Case Study-I: Detecting Flush+Reload on RSA

In this section, we present our first case study on detection of Flush+Reload attack on RSA cryptosystem.

*1) Detection Accuracy:* Detection accuracy is one of the primary indicators for evaluating a SCA detection framework. We use percentage accuracy to show the validity of trained machine learning models as we have used the same number of *no-attack* and *attack* samples in the training and validation data (i.e. attack and no-attack samples are not biased). Results in Table I show the achieved accuracy of the selected machine learning models. All four machine learning models show very high and consistent accuracy under all load conditions. Even under HL condition, the accuracy of LDA, LR and QDA stays above 99% while SVM shows above 95% accuracy. Most of the existing state-of-the-art detection mechanisms detect CSCAs in ZL conditions. Our results demonstrate that the proposed detection mechanism achieves very high accuracy for Flush+Reload attack under realistic load conditions. The primary reason behind this good accuracy of machine learning models can be explained with the help of Figures 5 and 6. Figures 5 and 6 illustrate the variation in magnitude of hardware events used for detection (also mentioned in Section IV-B) under attack and no-attack scenarios for ZL and HL conditions. Measurements show that all the features show clearly distinctive behavior under ZL. Under HL, apart from total number of execution cycles, the other features still maintain distinctive behavior leading to good performance of machine learning models.

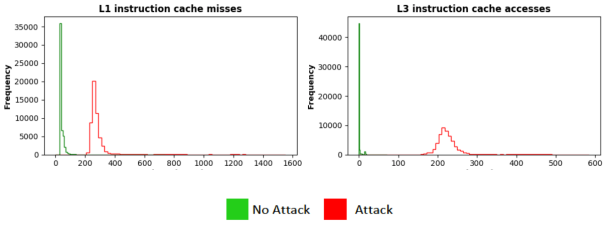


Fig. 5: Experimental results on 2 selected hardware events under ZL conditions for RSA encryption: With & Without Flush+Reload Attack

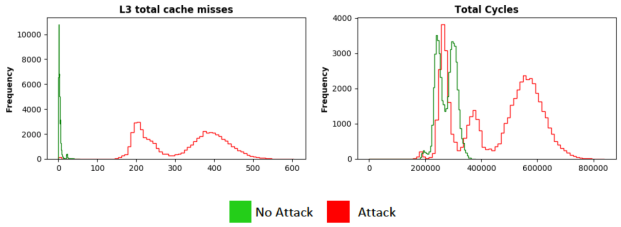


Fig. 6: Experimental results on 2 selected hardware events under HL conditions for RSA encryption: With & Without Flush+Reload Attack.

*2) Detection Speed:* Another important indicator to evaluate any run-time intrusion detection mechanism is the detection speed. Detection speed is an indirect reflection of how aggressively the detector profiles hardware events, which also affects the resultant performance overhead of the detection mechanism. Detection speed is a trade-off between how quickly an attack can be detected and how much overhead detection would cost. Flush+Reload is a single encryption attack [1]. For Flush+Reload, we consider detection speed as percentage of bits that are encrypted before the attack is successfully detected by our proposed detection mechanism. Various attacks [3], [30] have demonstrated that theoretically it is sufficient to retrieve 50% bits of secret key for a successful attack and other 50% bits of secret key can be reverse-engineered. Thus, a safe upper bound on the detection speed would be the detection before encryption of 50% of the bits of secret key i.e. before the encryption of 512 bits out 1024 bits in our case. As shown in Table I, all machine learning models used by the proposed detection mechanism are able to detect the attack way before this safe-bound. In all cases, the detection mechanism is able to detect Flush+Reload attack in the first 20 bits out of 1024 bits of RSA's execution.

*3) Confusion Matrix:* Detection of SCAs is a binary classification problem i.e. attack or no-attack. Therefore, detection inaccuracy can be further divided into false positives (cases when a no-attack condition is detected as an attack) and false negatives (cases when an attack condition is detected as no-attack) to analyze detection results in detail. Table I shows false positives and negatives exhibited by all machine learning models while detecting Flush+Reload on RSA. As shown in Table I, for LDA, LR and QDA majority of the miss-classifications belong to FPs, which can be considered less dangerous than FNs. In case of SVM, the behavior is different, exhibits more FN compared to FP under ZL and HL conditions.

*4) Performance Overhead:* Performance degradation is another key aspect to judge the applicability of detection mechanisms in real-time systems. In Section V-A2, it has been discussed that the detection granularity defines how efficiently the detection mechanism profiles hardware events and makes detection decisions which influence the performance of victim processes. Our proposed detection mechanism incurs $1-2\%$ performance degradation to the victim process while run-time profiling and detection mechanism is active. These results are achieved with the highest sampling frequency of performance

events. With the reduced sampling frequency, the performance overhead can be further reduced.

TABLE I: Results using LDA, LR, SVM & QDA models for Flush+Reload attack detection with RSA

| Model | Loads | Accuracy (%) | Speed (%) | FP (%) | FN (%) | Overhead (%) |
|---|---|---|---|---|---|---|
| LDA | ZL | 99.5 | 0.9 | .498 | .002 | 0.9 |
| | ML | 99.5 | 0.9 | 0.49 | .01 | |
| | HL | 99.4 | 0.9 | .527 | .073 | |
| LR | ZL | 99.5 | 0.9 | 0.5 | 0 | 1.6 |
| | ML | 99.5 | 0.9 | .494 | .006 | |
| | HL | 99.5 | 0.9 | .462 | .038 | |
| SVM | ZL | 98.8 | 0.9 | 0.4 | .78 | 1.3 |
| | ML | 90 | 0.9 | 0.17 | 9.83 | |
| | HL | 95.8 | 0.9 | 3.21 | .99 | |
| QDA | ZL | 99.5 | 0.9 | 0.5 | 0 | 0.6 |
| | ML | 99.5 | 0.9 | .494 | .006 | |
| | HL | 99.4 | 0.9 | 0.57 | .03 | |

TABLE II: Results using LDA, LR, SVM & QDA models for Flush+Reload attack detection with AES

| Model | Loads | Accuracy (%) | Speed (%) | FP (%) | FN (%) | Overhead (%) |
|---|---|---|---|---|---|---|
| LDA | ZL | 99.8 | 20 | 0.06 | 0.14 | 7.8 |
| | ML | 93.9 | 20 | 6.1 | .018 | |
| | HL | 91.5 | 20 | 6.3 | 2.2 | |
| LR | ZL | 99.9 | 40 | 0.1 | 0 | 2.7 |
| | ML | 88.4 | 40 | 11.6 | 0 | |
| | HL | 96.8 | 40 | 3.16 | 0.04 | |
| SVM | ZL | 99.9 | 20 | 0.1 | 0 | 3.9 |
| | ML | 88.5 | 20 | 11.5 | 0 | |
| | HL | 96.7 | 20 | 3.25 | .05 | |
| QDA | ZL | 99.6 | 20 | 0.22 | 0.18 | 8.3 |
| | ML | 93.8 | 20 | 6.13 | .07 | |
| | HL | 91.5 | 20 | 5.9 | 2.6 | |

### *B. Case Study-II: Detecting Flush+Reload on AES*

*1) Detection Accuracy:* Table II shows the achieved accuracy of machine learning models under different system conditions for Flush+Reload attack with AES. Under ZL condition all models are able to show very high accuracy (above 99%). The accuracy decreases as system load is increased. However, LR and SVM are still able to show above 96% accuracy under HL conditions.

*2) Detection Speed:* For Flush+Reload attack on AES [12], we sample performance counters after every 10 encryptions. Here, the detection speed is defined in terms of encryptions by which the attack is detected taken as a percentage of total 250 encryptions, which is the number of encyprtions an attacker performs to complete the attack. For example, a detection speed of 20% means that the attack was detected by the first 50 encryptions. As shown in Table II, all machine learning models except LR are able to detect attack in all cases by first 50 encryptions. In case of LR, the detection is achieved by first 100 encryptions, i.e., below 50% of 250 encryptions).

*3) Confusion Matrix:* Table II shows the distribution of inaccuracy shown by all machine learning models into false positives and false negatives while detecting Flush+Reload attack on AES. It can be observed that for almost all cases, the majority of the inaccuracies shown by all machine learning models fall into the category of false positives.

*4) Performance Overhead:* The performance overhead of run-time detection for all machine learning models is shown in Table II. LDA and QDA models show slightly high overhead, while the other two models exhibit a reasonable performance overhead. The primary reason for a relatively high overhead for detection of Flush+Reload attack on AES is high resolution sampling of performance counters which is necessary to detect Flush+Reload attack on AES before significant security degradation occurs (i.e. before 50 encryptions).

### *C. Case Study-III: Detecting Flush+Flush on 2 different Implementations of AES*

Most of the existing CSCA research works have not experimented with the attacks like Flush+Flush due to its stealth and non-detectable nature. According to [2], it is virtually impossible to detect the thread responsible for Flush+Flush attack due to absence of any abnormality in cache misses and hits for the attacker process. However, this does not stop from detecting the presence of a Flush+Flush attack as victim process results into more cache misses and accesses because of high speed flushing from the attacker process.

TABLE III: Results using LDA, LR, SVM & QDA models for Flush+Flush attack (FF_Imp1) detection

| Model | Loads | Accuracy (%) | Speed (%) | FP (%) | FN (%) | Overhead (%) |
|---|---|---|---|---|---|---|
| LDA | ZL | 99.9 | 25 | .075 | .025 | 1.18 |
| | ML | 98.7 | 25 | 1.16 | 0.14 | |
| | HL | 95.2 | 12.5 | 4.57 | 0.23 | |
| LR | ZL | 91.7 | 12.5 | 0 | 8.3 | 1.10 |
| | ML | 83.1 | 25 | 14.3 | 2.7 | |
| | HL | 75.9 | 25 | 24 | 0.1 | |
| SVM | ZL | 97.4 | 12.5 | 0 | 2.6 | 0.8 |
| | ML | 70.6 | 12.5 | 27.8 | 1.60 | |
| | HL | 63.2 | 12.5 | 36 | 0.8 | |
| QDA | ZL | 99.9 | 12.5 | 0.09 | 0.01 | 1.2 |
| | ML | 98 | 12.5 | 1.99 | .008 | |
| | HL | 91.1 | 12.5 | 8.85 | 0.05 | |

TABLE IV: Results using LDA, LR, SVM & QDA models for Flush+Flush attack (FF_Imp2) detection

| Model | Loads | Accuracy (%) | Speed (%) | FP (%) | FN (%) | Overhead (%) |
|---|---|---|---|---|---|---|
| LDA | ZL | 99.8 | 0.2 | .042 | .158 | 3.5 |
| | ML | 98.2 | 0.2 | 1.4 | 0.40 | |
| | HL | 80.2 | 0.1 | 8.2 | 11.6 | |
| LR | ZL | 88.8 | 0.3 | 2.2 | 9 | 3.4 |
| | ML | 86.8 | 0.4 | 5.9 | 7.3 | |
| | HL | 76.5 | 0.8 | 5.9 | 17.6 | |
| SVM | ZL | 85.2 | 0.1 | 14.2 | 0.6 | 3.7 |
| | ML | 73.3 | 0.1 | 25.4 | 1.3 | |
| | HL | 66.7 | 0.8 | 19.6 | 13.7 | |
| QDA | ZL | 89.7 | 0.2 | 10.1 | 0.2 | 4.5 |
| | ML | 82.1 | 0.2 | 17.1 | 0.80 | |
| | HL | 69.1 | 0.8 | 15.1 | 15.8 | |

*1) Detection Accuracy:* Table III and IV show the detection accuracy of all machine learning models for FF_Imp1 and FF_Imp2 of Flush+Flush attack. LDA and QDA show very high accuracy under all load conditions for detection of FF_Imp1 of Flush+Flush attack on AES. The high inaccuracy of LR

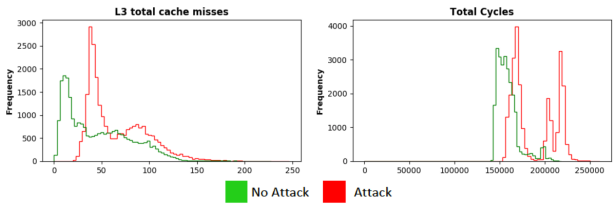


Fig. 7: Selected hardware events under HL conditions for AES encryption: With & Without Flush+Flush (FF_Imp1) Attack.

and SVM models under HL conditions can be explained with the help of Figure 7, which show the behavior of used hardware performance counters under attack and no-attack for HL conditions. In case of HL condition, it is evident that all the features start to overlap under attack and no-attack scenarios as shown in Figure 7. This behavior of overlapping features makes it harder for machine learning models to properly discern attack scenario from no-attack scenario. However, it is interesting to see that the LDA and QDA models are still able to show good accuracy in case of HL condition (95.20%). Similar kind of results are observed for FF_Imp2 of Flush+Flush attack. LDA shows the highest detection accuracy in comparison to the other models as shown in Table IV.

*2) Detection Speed:* The first implementation (FF_Imp1) of Flush+Flush on AES [2], needs to perform at least 350-400 encryptions whereas, the second faster implementation (FF_Imp2) of Flush+Flush on AES (modified from [12]), takes around 50,000 encryptions to complete the attack. Thus, in both cases, a detection of Flush+Flush attack would only be useful if it is made before the completion of 400 (FF_Imp1) and 50,000 (FF_Imp2) encryptions of AES. So, for Flush+Flush attack on AES, the detection speed is defined in terms of number of encryptions needed to detect the attack taken as a percentage of 400 (FF_Imp1) and 50,000 (FF_Imp2) encryptions (upper bound). As an example, a detection speed of 12.5% (for FF_Imp1) would mean that detection is achieved by first 50 encryptions. Whereas, a detection speed of 0.2% (for FF_Imp2) would mean that detection is achieved by first 100 encryptions. Both implementations of Flush+Flush attack on AES are shown in Table III and IV. The first implementation (FF_Imp1) of Flush+Flush attack is detected by first 50 encryptions in most of the cases. Detection is achieved by 100 encryptions in the other cases. The second implementation (FF_Imp2) of Flush+Flush attack is detected by first 100 encryptions in most of the cases and at maximum the attack is detected by first 400 encryptions which is far below 50% completion of total 50,000 encryptions.

*3) Confusion Matrix:* Table III and IV show breakdown of miss-classifications of all machine learning models into FPs and FNs while detecting both implementations of Flush+Flush attack on AES. For first implementation (FF_Imp1) of Flush+Flush on AES, for most of the cases the majority of mispredictions falls into FPs. A few cases (SVM and LR under ZL) where majority of errors falls into false negatives category, have very high accuracy and the actual number of false negatives and positives for them is very low. Similar is the case for the second implementation (FF_Imp2) of Flush+Flush attack on AES where most of the miss-classifications belong to FNs category.

*4) Performance Overhead:* All four machine learning models incur small profiling and detection overhead for both implementations of Flush+Flush attack as shown in Table III and IV. FF_Imp1 in Table III shows maximum overhead of 1.18 in the case of LDA while, FF_Imp2 in table IV shows degradation of maximum 4.5% in case of QDA, which is still reasonable.

## VI. Comparative Analysis with State-of-the-art

Table V provides a comparative analysis of our proposed detection technique with state-of-the-art. We provide this comparison with respect to the detection accuracy, speed, performance overhead and system load conditions in order to provide the reader with a perspective related to the existing techniques. Details on these techniques are already discussed in Section II-B. Authors in [23] detect Flush+Reload at an accuracy of F-score=0.93 before the 1/5th of completion of the attack in presence of noise in the background. F-score is a measure of accuracy in statistical analysis of binary classification. This technique, however, doesnot discusses the impact on performance degradation. The *HexPADS* [24] claims to have detected Flush+Reload and Prime+Probe attacks at 100% accuracy with 2% overhead. These results are reported under No load conditions, which may lead to erronous accuracy and increased overhead under noisy system load conditions. There is no discussion of how fast *HexPADS* can detect any of the tested attacks. The *CloudRadar* [16] claims to have detected Prime+Probe and Flush+Reload attacks at 100% accuracy and 5% overhead under No Load conditions as well. This technique also suffers from the same issues as *HexPADS*. Techniques proposed in [25] detects Flush+Reload and Prime+Probe attacks at 97% and 98% accuracy, respectively, and within 2% detection speed in presence of background noise but did not discuss the overhead of by proposed mechanism. Technique proposed in [34] claims to detect template attacks with 99% accuracy but does not provide detection speed, overhead and any variations in system load conditions. Technique proposed in [26] claims to work 100% accurately on Prime+Probe attack with 2% overhead. Authors, however, do not provide any results on the detection speed. Also, the percentage of false negatives increase under load conditions for the proposed technique. Technique proposed in [27] claims to detect Flush+Reload at 100% accuracy in isolated conditions and there is no discussion on the impact of detection speed and overhead under load conditions. Similarly, The *SCADET* technique proposed in [28] claims to perform 100% accurately under different load conditions but detection speed and performance overhead of this mechanism is not discussed. Whereas, it raises false alarms in load conditions and trace timing is long in some cases, which is not suitable for early stage detection.

Based on the comparison provided in Table V, our proposed detection mechanism evidently performs better under stringent

evaluation metrics. Our mechanism works for a larger set of attacks while performing run-time detection. We have provided results for four attacks on different cryptosystems, namely: Flush+ Reload on RSA, Flush+Reload on AES, Flush+Flush on AES half-key (FF_Imp1) and Flush+Flush on AES full-key (FF_Imp2). Reported accuracy of our mechanism for these attacks is 99.51%, 99.99%, 99.99% and 99.8%, respectively. Similarly, the detection speed is reported as $< 0.9\%$, $< 40\%$, $< 25\%$ and $< 0.8\%$, respectively. Moreover, under any load conditions, the performance overhead of our proposed mechanism remains $< 8\%$.

TABLE V: Comparative analysis of proposed detection mechanism with state-of-the-art [Note: NA refers to *Not Available*].

| Ref. | Attack | Accuracy | Speed | Overhead | Load/Noise |
|---|---|---|---|---|---|
| [23] | Flush+Reload | F-score 0.93 | 1/5th of attack completion | NA | Yes (Apache) |
| [24] | Flush+Reload Prime+Probe | 100% | NA | $< 2\%$ | No |
| [16] | Prime+Probe Flush+Reload | 100% | NA w.r.t to key completion | $< 5\%$ | No |
| [25] | Flush+Reload Prime+Probe | 97%, 98% | 2% | NA | Yes (SPEC) |
| [34] | Template Attacks | >99% | NA | NA | No |
| [26] | Prime+Probe | 100% | NA | 2% | Yes (CIW) |
| [27] | Flush+Reload | 100% | NA | NA | No |
| [28] | Prime+Probe | 100% | NA | NA | Yes (Open source) |
| Our Results | Flush+Reload (RSA) Flush+Reload (AES) Flush+Flush (AES, FF_Imp1) Flush+Flush (AES, FF_Imp2) | 99.51%, 99.99%, 99.99%, 99.8% | 0.9%, <40%, <25%, <0.8% | $< 8\%$ | Yes (Memory Intensive SPEC benchmarks) |

## VII. Conclusions

This paper presents a novel run-time detection mechanism for access-driven CSCAs on Intel's x86 architecture. The proposed detection mechanism is capable of detecting a large set of CSCAs with considerably high detection accuracy & speed while incurring minimum performance overhead under variable system load conditions. The paper provides experimental validation on 04 state-of-the-art CSCAs under stringent evaluation metrics. Our results show detection accuracy up to 99.51% for Flush+Reload attack on RSA, incurring a worst-case performance overhead of only 1.63% and 99.99% accuracy on AES while incurring a performance overhead of 8.28% under noisy system load. For Flush+Flush attack, the detection accuracy of up to 99.97% and 99.85% is achieved for two different implementations of attack on AES with a performance overhead of 1.18% & 3.51%, respectively. The novelty in this work lies in the fact that we were able to detect high-resolution attacks, such as Flush+Reload, and stealthy attacks, such as Flush+Flush, on different cryptosystems under realistic system load conditions using machine learning models with selected hardware events.

## VIII. Acknowledgements

This research work is partially supported by PHC PERIDOT project e-health.SECURE (Pak-France) and NCCS, Pakistan.

## References

[1] Y. Yarom and K. Falkner, "FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack," in *USENIX Security 14*, p. 719.
[2] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+Flush: A fast and stealthy cache attack," in *DIMVA*, 2016, pp. 279–299.
[3] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: The case of AES," *CT-RSA*, pp. 1–20, 2006.
[4] 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.
[5] S. A. et al, "Cross-VM cache-based side channel attacks and proposed prevention mechanisms: A survey," *Journal of Network and Computer Applications*, pp. 259 – 279, 2017.
[6] X. Jin, H. Chen, X. Wang, Z. Wang, X. Wen, Y. Luo, and X. Li, "A simple cache partitioning approach in a virtualized environment," in *IEEE ISPA*, Aug 2009, p. 519.
[7] H. Raj, R. Nathuji, A. Singh, and P. England, "Resource management for isolation enhanced cloud services," in *CCSW*, 2009, pp. 77–84.
[8] F. Liu and R. B. Lee, "Random fill cache architecture," in *MICRO*.
[9] T. Kim, M. Peinado, and G. Mainar-Ruiz, "STEALTHMEM: System-level protection against cache-based side channel attacks in the cloud," in *USENIX Conference on Security Symposium*, 2012, pp. 11–11.
[10] P. K. et al, "Spectre attacks: Exploiting speculative execution," 2018.
[11] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown."
[12] G. Irazoqui, M. S. Inci, T. Eisenbarth, and B. Sunar, "Wait a minute! a fast, cross-VM attack on AES," in *RAID 17*.
[13] M. S. Inci, B. Gulmezoglu, G. Irazoqui, T. Eisenbarth, and B. Sunar, "Cache attacks enable bulk key recovery on the cloud." CHES, 2016.
[14] D. Gruss, R. Spreitzer, and S. Mangard, "Cache template attacks: Automating attacks on inclusive last-level caches," in *USENIX*, 2015.
[15] M. Godfrey and M. Zulkernine, "Preventing cache-based side-channel attacks in a cloud environment," *IEEE TCC*, vol. 2, no. 4, 2014.
[16] T. Zhang, Y. Zhang, and R. B. Lee, "CloudRadar: A real-time side-channel attack detection system in clouds," in *RAID*, 2016.
[17] Z. Zhou, M. K. Reiter, and Y. Zhang, "A software approach to defeating side channels in last-level caches," in *ACM SIGSAC*, 2016.
[18] Z. Wang and R. B. Lee, "New cache designs for thwarting software cache-based side channel attacks," in *ISCA*, 2007, pp. 494–505.
[19] D. Cock, Q. Ge, T. Murray, and G. Heiser, "The last mile: An empirical study of timing channels on seL4," in *ACM SIGSAC*, 2014.
[20] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Cross-VM side channels and their use to extract private keys," in *ACM CCS*, 2012.
[21] M. Godfrey and M. Zulkernine, "A server-side solution to cache-based side-channel attacks in the cloud," in *IEEE CLOUD*, 2013.
[22] D. Gruss, C. Maurice, A. Fogh, M. Lipp, and S. Mangard, "Prefetch side-channel attacks: Bypassing smap and kernel ASLR," in *ACM CCS*.
[23] M. Chiappetta, E. Savas, and C. Yilmaz, "Real time detection of cache-based side-channel attacks using hardware performance counters," *Appl. Soft Comput.*, 2016.
[24] M. Payer, "HexPADS: a platform to detect "stealth" attacks," in *ESSoS*, 2016.
[25] Z. Allaf, M. Adda, and A. Gegov, "A comparison study on Flush+Reload and Prime+Probe attacks on AES using machine learning approaches," *UKCI*, 2017.
[26] 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 *FMEC*, 2018.
[27] S.-h. PENG, Q.-f. ZHOU, and J.-l. ZHAO, "Detection of cache-based side channel attack based on performance counters," *DEStech*, 2017.
[28] M. Sabbagh, Y. Fei, T. Wahl, and A. A. Ding, "SCADET: a side-channel attack detection tool for tracking Prime+ Probe," in *ICCAD*, 2018.
[29] D. Gullasch, E. Bangerter, and S. Krenn, "Cache Games – bringing access-based cache attacks on AES to practice," in *IEEE SSP*, 2011.
[30] Y. Yarom, D. Genkin, and N. Heninger, "CacheBleed: a timing attack on OpenSSL constant-time RSA," *Journal of Crypt. Engg. 2017*.
[31] "Performance application programming interface," 2018, http://icl.cs.utk.edu/papi/.
[32] C. M. Bishop, *Pattern Recognition and Machine Learning*, 2006.

[33] “https://www.spec.org/benchmarks.html,” 2018.
[34] M. A. et al, “Performance counters to rescue: A machine learning based safeguard against micro-architectural side-channel-attacks,” Crypt., 2017.