WHISPER A Tool for Run-time Detection of Side-Channel Attacks

Maria Mushtaq, Jeremy Bricq, Muhammad Khurram Bhatti, Ayaz Akram, Vianney Lapotre, Guy Gogniat, Pascal Benoit

IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

WHISPER
A Tool for Run-time Detection of Side-Channel Attacks

MARIA MUSHTAQ[1], JEREMY BRICQ[2], MUHAMMAD KHURRAM BHATTI[3], AYAZ AKRAM[4], VIANNEY LAPOTRE[5], GUY GOGNIAT[5], AND PASCAL BENOIT[1]
[1]LIRMM, Univ Montpellier, CNRS, Montpellier, France
[2]University of Brussels, Brussels, Belgium
[3]ECLab, Information Technology University, Lahore, Pakistan
[4]University of California Davis, CA, USA
[5]Lab-STICC, Université Bretagne Sud, Lorient, France

Corresponding author: Maria Mushtaq (e-mail: maria.mushtaq@lirmm.fr).

This work is partially supported by the NCCS and PHC PERIDOT Project e-health.SECURE (Grant ID: 3-6/HEC/R&D/PERIDOT/2017).

ABSTRACT

High resolution and stealthy attacks and their variants such as Flush+Reload, Flush+Flush, Prime+Probe, Spectre and Meltdown have completely exposed the vulnerabilities in Intel's computing architecture over the past few years. Mitigation techniques against such attacks are not very effective for two reasons: 1) Most mitigation techniques protect against a specific vulnerability and do not take a system-wide approach, and 2) they either completely remove or greatly reduce the performance benefits of resource sharing. In this work, we argue in favor of detection-based protection, which would help apply mitigation only after successful detection of the attack at runtime. As such, detection would serve as the first line of defense against such attacks. However, for a detection based protection strategy to be effective, detection needs to be highly accurate, to incur minimum system overhead at runtime, should cover a large set of attacks and be capable of early stage detection, *i.e.*, at the very least before the attack is completed. We propose a machine learning based side-channel attack (SCA) detection tool, called WHISPER that satisfies the above mentioned design constraints. WHISPER uses multiple machine learning models in an Ensemble fashion to detect SCAs at runtime using behavioral data of concurrent processes, that are collected through hardware performance counters (HPCs). Through extensive experiments with different variants of state-of-the-art attacks, we demonstrate that the proposed tool is capable of detecting a large set of known attacks that target both computational and storage parts in computing systems. We present experimental evaluation of WHISPER against Flush+Reload, Flush+Flush, Prime+Probe, Spectre and Meltdown attacks. The results are provided under variable system load conditions and stringent evaluation metrics comprising detection accuracy, speed, system-wide performance overhead and distribution of error (*i.e.*, False Positives & False Negatives). Our experiments show that WHISPER can detect a large and diverse attack vector with more than 99% accuracy at a reasonably low performance overhead.

INDEX TERMS Side-Channel Attacks (SCAs), Cryptography, Detection, Machine Learning, Security, Privacy.

I. INTRODUCTION

Information security is fast becoming a first-class design constraint in almost all domains of computing. Modern cryptographic algorithms are used to protect information at the software level. These algorithms are theoretically sound and require enormous computing power to break with brute-force. For instance, for a 128-bit AES key, it would take 5.4×10^{18} years to crack the AES using a computer capable of performing 10^6 decryption operations per μs [1]. However, recent research has shown that cryptosystems, including AES, can be compromised due to the vulnerabilities of the underlying hardware on which they run. Side-channel attacks exploit such physical vulnerabilities by targeting the platforms on which these cryptosystems execute [2].SCAs can use a variety of physical parameters, *e.g.*, power consumption, electromagnetic radiation, memory accesses and timing patterns to extract secret keys/information [3], [4], [5], [6]. The baseline idea here is that the SCAs can exploit variations in these parameters during the execution of cryptosystems on a particular hardware and can determine the secret information used by cryptosystems based on the observed parameters. Cache-based side-channel attacks (CSCAs) are a special type of SCA, in which a malicious process deduces the secret information of a victim process by observing its use of caching hardware. Such attacks often rely on the presence of specialized instructions to manipulate the state of shared caches. The inherent features that any known CSCA exploits are cache timing and access patterns. In order to make an attack possible, an attacker usually needs to identify the victim's memory addresses by observing the shared caches. Further, an attacker would also have to determine if a particular memory access generated by the victim results in a cache hit or a miss. Generally, attackers observe the victim's memory accesses indirectly rather than directly. Thus, they change the usual utilization (with respect to a no-attack scenario) of underlying hardware resources, particularly caches. Such attacks can be prevented at various levels including at the system level, hardware level and application level [7]. At the system level, physical and logical isolation approaches exist [8]. At the hardware level, mitigation techniques are rather difficult due to the cost and complexity of their design. However, hardware solutions suggest having new secure caches, changes in prefetching policies and either randomization or complete removal of cache interference [9]. At the application level, the proposed countermeasures tend to target the source of information leakage and mitigate it [10]. The attacks are becoming sophisticated and stealthier [4], [7]. They overcome statically applied mitigation techniques. Therefore, on the one hand, protection against these CSCAs needs to be applied across the entire computing stake and, on the other hand, mitigation strategies must not reduce or remove the hard-earned performance benefits of computing systems. There is a niche for using detection-based protection, which would help apply mitigation only after successful detection of CSCAs.

This work addresses the problem of accurate and early detection of CSCAs at run-time. In this paper, we propose a machine learning system based on a runtime detection tool, called WHISPER, for CSCAs targeting Intel's x86 architecture. We demonstrate that intelligent performance monitoring of concurrent processes at hardware level, coupled with machine learning methods, can enable early detection of high precision and stealthier CSCAs. The state-of-the-art, discussed in Section II, suggests that some solutions exist based on machine learning for detection of CSCAs such as [11], [12], [13], [14], [15]. However, there are three major limitations in the prior work. The first is that the majority of machine learning models used in these solutions are trained to classify a specific attack, or a subset of attacks, belonging to any one category. Thus, when exposed to other CSCAs, these models have to be retrained. The second limitation is that, even after the machine learning model is retrained, it may not yield the same accuracy because different CSCAs exploit different cache vulnerabilities and the same model might simply not be capable of accurately classifying the changed behavior. And the third limitation is, prior works that are capable of detecting different attack techniques together are proved to have ill-suited evaluation criteria, *i.e.*, they do not account for performance overhead, detection speed, miss classification rates and load conditions simultaneously, for accurate and fast detection of attack behaviors at runtime. In practice, the system can be exposed to multiple attacks of different categories, in any temporal order and under any system load conditions. Thus, retraining or changing individual machine learning models may not be feasible, particularly for runtime detection tools. A more generic solution is needed that takes into account all (or most) attack techniques and, at the same time, well-suited evaluation criteria for detection.

In this paper, we use multiple machine learning models in an ensemble fashion in the WHISPER tool. The use of the ensemble model, instead of an individual machine learning model, is motivated by the fact that the proposed tool is designed to detect a large set of CSCAs. The ensemble model uses behavioral data of concurrent processes running on Intel's x86 architecture. These data are collected from different hardware events using HPCs, in near real-time, and are used as features. These data represent the pattern of memory accesses generated by data-dependent cryptographic operations that are being carried out by the underlying hardware. Since each CSCA generates a different interference with caches, the data being captured through HPCs at run-time can therefore lead to miss-classification for a single machine learning model. Instead an Ensemble model, incorporates multiple best-performing models and performs a majority-vote before classifying a given situation as Attack or No-Attack. It is thus capable of accurately detecting a larger set of attacks. Through extensive experiments, our goal is to evaluate the capability of the proposed tool to detect different variants of three state-of-the-art CSCAs, namely; Flush+Reload, Flush+Flush and Prime+Probe. The experiments are conducted to illustrate the proposed tool's capability of detecting almost all major known attack categories that are based on cache access patterns. The following are the main contributions of this work.

1) The paper proposes a run-time detection tool to detect CSCAs. The tool, called WHISPER, uses multiple machine learning models in an ensemble fashion and relies on the run-time profiling of concurrent processes that are collected directly through the hardware events using HPCs in near real time.
2) The paper demonstrates the capability of the proposed tool to detect access-driven side-channel attacks with reasonably high detection accuracy, high detection speed, low performance overhead and minimal false

positives and negatives.

3) For validation purposes, the paper presents experimental evaluation of the proposed tool against a large and diverse attack vector comprising both computational and storage attacks. Results for the detection of state-of-the-art CSCAs, such as Flush+Reload, Flush+Flush and Prime+Probe are presented and discussed. Results on the detection of computational attacks, such as Spectre and Meltdown are also presented to demonstrate the scalability of WHISPER tool.
4) We demonstrate that the WHISPER tool is resilient to noise generated by the system under various loads. To do so, we provide results under realistic system load conditions, *i.e.*, under No Load (NL), Average Load (AL) and Full Load (FL) conditions. These load conditions are achieved by concurrently running memory-intensive SPEC benchmarks on the system along with the cryptosystem and attacks.
5) We provide detailed discussion, supported with experimental data, about the selection of appropriate machine learning models and hardware events for run-time detection of CSCAs. Our results show that the proposed tool can also be used for other attacks (*i.e.*., Spectre and Meltdown attacks) without requiring any changes in the selected HPCs or retraining of classifiers.

The rest of the paper is organized as follows. Section II, provides the state-of-the-art on CSCAs, their mitigation and detection techniques. Section III discusses selected attacks as use-cases for evaluation of the proposed tool. Section IV presents the WHISPER tool, its components and design challenges associated with detection. Section V provides experimental evaluation using recent attacks from 3 CSCA categories. Section VI provides additional results on the detection of computational attacks to showcase the scalability of the WHISPER tool. Section VII discusses the results and Section VIII presents concluding remarks on our findings.

II. RELATED WORKS

In this section, our main aim is to provide a detailed background on the state-of-the-art related to CSCAs, their mitigation techniques and their proposed detection mechanisms.

A. STATE-OF-THE-ART ON CACHE SIDE-CHANNEL ATTACKS AND MITIGATION TECHNIQUES

CSCAs are a significant class of SCAs that exploit the execution-level details of cryptosystems by observing their interference with the cache hierarchy. The main sources of information leakage exploited by CSCAs are the memory access pattern of target cryptosystem and the access timing disparity in cache hierarchy. Many side- and covert-channel attacks have already been proposed [16]. For instance, some cache-based side-channel attacks are Flush+Reload [3], Flush+Flush [4], Prime+Probe [17], Evict & Time [5] and Evict & Reload [18]. Similarly, some covert-channel attacks include: Foreshadow [19] and May the Fourth Be With You [20]. More recently, new vulnerabilities have been discovered that exploit computational optimizations such as Out-of-Order and Speculative execution. Spectre [21] and Meltdown [22] attacks are the most recent examples of such attacks. Both Spectre and Meltdown are two-phase attacks and, in one of their phases, they require CSCAs to retrieve privileged information from caches. CSCAs are broadly divided into two classes, *i.e.*, time-driven and trace-driven attacks. Since cache hits and misses exhibit different timing for the same operation, this difference allows time-driven attacks to extract information on the secret key. Trace-driven attacks, on the other hand, try to access the cache lines that are being used by the victim by analyzing the cache state.

Multiple mitigation techniques have also been proposed against these CSCAs in the last decade. These techniques can be categorized into logical & physical isolation techniques, noise-based techniques, scheduler-based techniques and constant time techniques (referring to different cache levels in the cache hierarchy). For instance, logical physical isolation techniques include Cache Coloring [23], CloudRadar [24], STEALTHMEM [10], NewCache [25] and Hardware Partitioning [26]; noise-based techniques include fuzzy times [27], bystander workloads [28]; and scheduler-based techniques include obfuscation [28] and minimum timeslice [29]. Our Study of these mitigation solutions reveals that most techniques focus on only one specific vulnerability in the cache hierarchy. In practice, a system can be exposed to multiple attacks simultaneously or in any temporal order. Moreover, with relatively smarter attacks, these mitigation approaches can be bypassed. Moreover, researchers have discussed the fact that these attacks have not been completely patched to-date [7], [30]. Researchers have also debated that cryptographic code designers are far away from incorporating the appropriate counters to avoid cache leakages [31]. Since in this work, our focus is on the detection strategies of CSCAs, we provide more insight into detection mechanisms in the following and invite interested readers to explore further by following up references to mitigation techniques.

B. STATE-OF-THE-ART ON DETECTION MECHANISMS

SCA detection techniques are divided into two basic categories; signature-based and anomaly-based detection. Some techniques use a combined approach as well, *i.e.*, signature + anomaly-based detection. Signature-based techniques depend on signature of known SCAs. At run time, program execution is compared with an already generated signature and in the case of a match, an attack is detected. Such detection mechanisms show good accuracy for known attacks but suffer from low accuracy in the case of unknown or modified attacks [24]. Anomaly-based detection approaches generate a model for normal/benign applications. Any significant deviations from the model will be detected/considered as an attack. Anomaly-based detection mechanisms have the capability to detect unknown and modified attacks, but they can suffer from high false positives as it is difficult to include the behavior of every benign application. Also, benign applications can potentially resemble CSCAs due to intensive memory usage [24]. There are some detection mechanisms that merge both anomaly and signature based detection mechanisms to achieve accurate results [32], [24].

1) Signature-based Detection

Allaf *et al.* [15] propose a mechanism to inspect Prime+Probe and Flush+Reload attacks targeting AES cryptosystem. Their mechanism uses ML models and HPCs but the work lacks the basic evaluation criteria required for detection. The proposed mechanism shows good accuracy under isolated conditions, where the attacker and victim are the only load on the system whereas, accuracy degrades under realistic load conditions. Moreover, the authors do not discuss the impact on overall performance caused by the said technique. Mushtaq *et al.* [12] proposed a signature-based run-time detection mechanism called the NIGHTs-WATCH. It comprises 3 linear machine learning classifiers, LDA, LR, and SVM, that work independently. The NIGHTs-WATCH takes different hardware events as input features under attack/no-attack scenarios. The authors claim a low performance overhead for a high detection accuracy and speed. This work is limited to the use of individual ML models trained and used for specific attacks. This means that models need to be retrained for each new attack to detect it. In a similar work, Mushtaq *et al.* in [13] used both linear and non-linear ML classifiers to detect different implementations of Prime+Probe attacks running on the AES cryptosystem.

In [33], Sabbagh *et al.* proposed a signature-based detection tool, SCADET, that detects Prime+Probe attacks. Instead of using HPCs, this approach uses high-level semantics and invariant patterns of attack, *i.e.*, I-cache, D-cache and LLC. Results show that SCADET provides very good accuracy, but no results on detection speed or on the performance overhead of the mechanism, which leaves the question of runtime adaptation. Authors report that, in some cases, the system provides false alarms under load conditions. Moreover, the trace analysis time in this approach is very long, meaning the solution is not suitable for runtime detection. Notable irregularities have also been reported when the trace exceeds a certain size.

Some of the signature-based detection mechanisms use threshold determination to detect CSCAs instead of machine learning [34], [35], [36]. One of the techniques in [34], named as HexPADS, utilizes the values of cache miss rates and page faults to detect an attack. HexPADS is able to detect cache template attacks [18] based on Flush+Reload, enhanced version of C5 attack [37] based on Prime+Probe, page fault attacks (CAIN [38]) and fault attacks based on Rowhammer [39]. However, the paper lacks a discussion on detection speed. Moreover, HexPADS assumes no variation in the system load, therefore, overhead may increase with realistic scenarios. Another similar threshold-based technique is presented in [35], which uses cache miss rate and data-TLB miss rate to recognize CSCAs. Authors reported results depicting successful detection of variants of Flush+Reload attacks. However, the authors mentioned neither performance overhead caused by this detection technique nor the detection speed. Raj and Dharanipragada [36] presented Pokerface to identify and mitigate CSCAs. Pokerface compares the memory bus bandwidth with a threshold level to detect Prime+Probe and Flush+Reload CSCAs. However, the authors do not provide a discussion of detection accuracy or mention the speed of the proposed mechanism. A threshold-based detection technique Deja-Vu [40] was introduced to detect attacks on programs guarded by SGX. This technique uses test benchmarks with a negligible overhead. However, the instrumentation required can increase the size of enclave binaries to a very large number and threshold determination is not always very sophisticated or provides a reliable approximation for detecting attacks.

2) Anomaly-based Detection

Recently, two new detection techniques that rely on anomaly-based detection have been proposed [41], [42]. CacheShield [41] is an anomaly-based detection mechanism for CSCAs on legacy software (victim applications) that monitors HPCs while using an unsupervised anomaly detection algorithm called Cumulative Sum Method (CUSUM). The CacheShield detects Flush+Reload, Flush+Flush and Prime+Probe attacks under load conditions and the results show that attacks were detected before the attack was 50% complete. Another anomaly-based detection mechanism is a semi supervised technique called the SpyDetector [42]. It has been proposed to detect CSCAs under variable load conditions using HPCs. The SpyDetector has been validated on Flush+Reload, Flush+Flush and Prime+Probe attacks running over RSA, AES and ECDSA cryptosystems. The mechanism performs well under variable load conditions on physical as well as on virtual setups, but the authors do not provide any results regarding detection speed and the overhead for reported accuracy.

3) Signature + Anomaly Based Detection

As discussed earlier, there are detection techniques that use a combination of both signature- and anomaly-based detection approaches [11], [24], [32]. Chiappetta *et al.* [11] proposed a machine learning based detection mechanism for Flush+Reload attack on AES and ECDSA cryptosystems. This work uses three approaches to detect the attack: correlation-based detection, anomaly detection and artificial neural networks-based detection. This detection mechanism offers good detection accuracy and speed but does not incur less overhead. Another signature and anomaly-based detection mechanism is CloudRadar [24], which correlates cryptographic execution of applications on virtual machines and the anomalous behavior of caches to detect CSCAs in cloud systems. The CloudRadar demonstrates its efficiency by performing an attack and once it is detected, VM migration is performed, which serves as a lightweight patch to cloud systems. The proposed mechanism is tested for Flush+Reload and Prime+Probe attacks and offers high accuracy and negligible overhead at a predetermined sampling frequency and threshold. This approach does not take machine learning into account for analyzing variable behaviors and performs a threshold-based decision which is unreliable for different attack behaviors at runtime, where, the realistic load conditions can be noisy and attack behavior may vary from predetermined threshold based behavior. Although *CloudRadar* detects attacks with high accuracy in isolated conditions, no tests have been performed under realistic/noisy scenarios to measure the efficiency of the detection mechanism. Another three-step detection mechanism on cache and branch predictor based CSCAs is proposed in [32]. This method includes HPCs and machine learning models and uses three stages; detecting the anomaly, finding the class of anomaly and correlating malicious process with the victim. Their experiments show that this mechanism performs well and provides high accuracy under a predetermined sampling frequency and a specific threshold, but the detection speed and detection overhead of the proposed technique are not evaluated or discussed.

Based on our findings, the overall state-of-the-art on detection techniques are lacking in certain aspects that are crucial for making run-time detection an effective way to mitigate SCA attacks. For instance, most of the proposed techniques primarily focus on achieving higher detection accuracy and either ignore or insufficiently cater for other parameters like detection speed and performance overhead. Moreover, the attack surface is continuously expanding, but the proposed techniques target only selected attacks and do not try to cover a larger attack vector. We believe that, for run-time detection to be an effective way of mitigating SCAs, the detection tools need to cover a large attack vector while offering reasonably good detection accuracy, speed and overhead.

III. USE-CASES: SELECTED CACHE SIDE CHANNEL ATTACKS

As highlighted in Section I, we evaluated the proposed tool against a large and diverse attack vector to demonstrate its adaptability. Our primary focus, however, is on cache side-channel attacks. We selected six different CSCA implementations as use-cases for the validation of the WHISPER tool. These attacks cover three main categories of CSCAs, *i.e.*, Flush+Reload (F+R), Prime+Probe (P+P) and Flush+Flush (F+F). In the state-of-the-art, these selected use-case attacks often target a given cryptosystem to retrieve secret key information. Therefore, for the purpose of uniformity and demonstration, all the experimental results in this work are mainly shown for one cryptosystem, *i.e.*, AES. However, it is worth mentioning here that the WHISPER tool also works effectively in other cryptosystems, such as RSA. For the selected CSCA use-cases, we used two different versions of OpenSSL on which the attacks are demonstrated in the state-of-the-art. Table 1 details these use-cases. Later, in Section VI, we demonstrate that the WHISPER tool is also capable of detecting attacks that are independent of the cryptosystem, such as Spectre and Meltdown attacks.

To serve the community at large, we provide the source code and experimental data related to all these CSCAs at our Github repository [43], which can be freely accessed, used, distributed and reproduced.

TABLE 1: List of Selected CSCAs as Use-Cases

No.	Use-case CSCAs	OpenSSL Version	Key Recovery
1	Flush+Reload	0.9.7l/ 1.0.1f	Half Key
2	Flush+Reload	0.9.7l/ 1.0.1f	Full Key
3	Flush+Flush	0.9.7l/ 1.0.1f	Half Key
4	Flush+Flush	0.9.7l/ 1.0.1f	Full Key
5	Prime+Probe	0.9.7l/ 1.0.1f	Half Key
6	Prime+Probe	0.9.7l/ 1.0.1f	Full Key

A. FLUSH+RELOAD

Flush+Reload [3] is a trace-driven attack that relies on the presence of page sharing. State-of-the-art attacks have been performed using the Flush+Reload technique using RSA and AES cryptosystems [44], [6]. This technique has three major steps; first the attacker flushes the shared cache line using CLFLUSH instruction. In the second step, the attacker lets the victim execute and in the third step, the attacker reloads the cache line and measures the reload time. Reload time indicates if the cache line was of interest for the victim or not. Two different variants of this attack were implemented on the AES cryptosystem for our test cases, which are available in [6], [45] (full key, faster implementation), [46] (half key, slower implementation).

B. PRIME+PROBE

Prime+Probe attacks are in the category of trace-driven attacks that exploit last level shared caches across multiple cores. The principle of Prime+Probe has been used to develop various CSCAs such as those proposed in [28], [47], [5], [44], [48], [49]. The Prime+Probe technique involves two major steps; in the first step, the attackers prime the cache with their own data and let the victim execute, in the second step, the attackers probe the cache and measure the time needed to access their own primed data in cache. The difference in time will inform the attacker if any of the cache set has been accessed by the victim or not. The Prime+Probe technique has been used at different cache levels including L1-Data cache (L1-D) [48], L1-Instruction cache (L1-I) [50] and Last Level Cache (LLC) [51]. Two different variants of this attack technique are available; one implementation contains half key recovery with a slower implementation [46], whereas we implemented the Flush+Reload on AES from [6], [45] and modified it for faster and full key recovery on the same principle for Prime+Probe.

C. FLUSH+FLUSH

Flush+Flush [4] attacks also belong to trace-driven attack techniques, but replace the Reload step of Flush+Reload with a Flush step. Attacks rely on CLFLUSH instructions to perform the Flush step. This technique has two major steps;

in the first step, the attacker flushes the cache line from the shared address space and lets the victim to operate normally (the victim may or may not access the flushed cache line). In the second step, the attacker flushes the same cache line and measures the time of flushing (the attacker determines if the victim has accessed that cache line or not during its normal execution). We implemented two different variants of this attack technique on the AES cryptosystem in our test cases and these are available on [4], [46] (this implementation targets the first round of encryption to recover half key which is rather slow). We also implemented Flush+Reload on the AES from [6], [45] and modified it for faster and full key recovery on the same principle as that used in Flush+Flush. The Flush+Flush attack is considered a stealthy, high-resolution and non-detectable CSCA. The Flush+Flush attack is also considered to be fast because unlike other CSCAs, it does not access any memory. Flush+Flush causes minimal cache hits and no cache miss at all due to constant flushes mechanism. Gruss *et al.* [4] claimed that the spy process in this attack can not be detected by monitoring cache behavior (hits, misses). Due to its working principle, a Flush+Flush attack causes more cache misses and memory accesses for the victim process due to constant flushing of its instruction/data.

We build our claim based on the fact that detection is required to indicate the presence or absence of intrusion from the victim's perspective. That is, for detection as a first step, it is important to identify when an intrusion is happening in the system in order to take the right protective measures. Then, the steps can be taken to identify which particular process in the system is the malicious one.

IV. WHISPER -A RUN-TIME SCA DETECTION TOOL

Having established the background and state-of-the-art in the previous sections, we now present the design details of our proposed CSCA detection tool, WHISPER. One of the distinguishing features of the WHISPER tool is that it is designed to work at runtime, *i.e.*, when the attack is actually happening, to detect and help the operating system protect itself against known CSCAs. The three challenges we address in designing the WHISPER tool are: 1) Detection tools usually approximate the whole system behavior which can increase the number of false positives and false negatives at runtime, 2) The detection process can slow down the overall execution of the cryptosystem, which can lead to a significant performance overhead while trying to achieve greater detection accuracy and 3) Detection can sometimes be very slow, resulting in late detection in the sense that the attacker has already completed up to 50% of its activity, for instance, secret key retrieval. In the literature, 50% is considered as a theoretical threshold for a successful attack [2], [5]. We considered all these design challenges as our evaluation metrics for the WHISPER tool.

The WHISPER tool does intelligent performance monitoring of concurrent processes with data collected at the hardware-level using hardware events and feeds them to selected machine learning models to perform early detection of high precision and stealthier CSCAs. The tool has two major components: 1) Selection of appropriate hardware events that will provide, at runtime, insight into the cache behavior while CSCAs take place and 2) Selection of appropriate machine learning methods that could perform binary classification of Attack vs No-Attack scenarios with high accuracy, high speed and minimum performance overhead.

We consider that the tool will operate under realistic system load conditions on commodity hardware. Therefore, we emulate the load conditions by running memory-intensive SPEC benchmarks on the system as background load. The load conditions are defined such that a No Load (NL) condition involves only victim and attacker processes running, an Average Load (AL) involves victim, attacker and any two SPEC benchmarks running, and a Full Load (FL) condition involves Victim, Attacker and any four SPEC benchmarks running in the background. In the state-of-the-art, it often happens that attacks are running in isolated conditions *i.e.*, attacker and victim are the only load. For instance, the attacks presented in [4], [5] and [3] have no load conditions. Therefore, to assume a realistic scenario, it is useful to validate the mechanism under different load conditions, which, in turn, validate the functioning of the tool in a realistic scenario. This is why we deliberately used SPEC benchmarks that are memory intensive and affect caches in terms of cache accesses, CPU cycles, cache hits and misses, *etc.*

In this section, we detail the methodology of the WHISPER tool and selection criteria for the two aforementioned components. We validate the tool experimentally on six different CSCAs and their variants. The CSCA use cases considered cover a large attack surface. In this work, we evaluate the efficacy of the proposed tool with these CSCAs being validated on different OpenSSL versions. We provide empirical evidence for the effectiveness of the proposed detection tool against these attacks and variants.

A. METHODOLOGY

Figure 1 is an abstract view of the methodology used in the WHISPER tool. We consider shared memory architecture as most known CSCAs target Intel's x86 based execution platforms. As illustrated, the tool collects behavioral data on concurrent processes at runtime using HPCs. As discussed in Section IV-B, these data comprise selected hardware events, which are fed to an Ensemble model. The WHISPER tool methodology consists of three distinct and significant phases, namely; 1) Run-time profiling, 2) Training machine learning models and 3) Classification & detection. In the following, we describe these phases in detail.

1) Run-time Profiling

The first phase of WHISPER's detection mechanism comprises runtime profiling of the victim's process *i.e.* target cryptosystem. In this phase, runtime samples from selected hardware events are collected using HPCs. Since we target access-driven CSCAs, we consider only the hardware events that are most affected by these attacks. Section IV-B provides

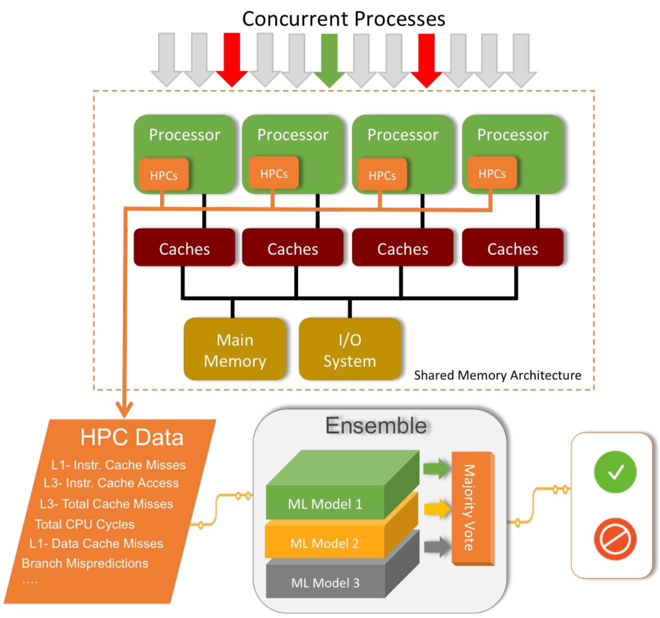

FIGURE 1: WHISPER Tool's Methodology-Abstract view

details on the selection of such events. Intel x86 processors [52] provide access to hundreds of hardware events through software APIs, which can reveal valuable information on the system.

WHISPER offers runtime profiling at variable sampling granularity. That is, the tool allows both the user and the system to adjust sampling granularity between fine-grain and coarse-grain levels, depending on the prevailing threat level. Sampling granularity is defined as the rate at which the data from HPCs are collected with respect to the completion of encryption. For example, Flush+Reload is a single encryption attack on RSA. Therefore, in this case, the sampling granularity would be after every $n-$ bits of encryption defined by the user. For attacks that require multiple encryptions to be completed, the sampling rate can be defined as $n-$ encryptions. For instance, Flush+Flush attack on AES requires between $350-400$ encryptions at least. Therefore, rather coarse-grain sampling would still be enough to detect this attack before its completion.

Sampling granularity has a major influence on the victim's performance as it varies the execution time of the victim's process and its shared libraries compared to normal execution. A higher sampling rate for hardware events would provide precise measurement of cache behavior, which would facilitate in faster detection speed. However, it will also slowdown the encryption/decryption process of the victim and increase the performance overhead. Therefore, sampling granularity implies a trade-off between detection speed and performance overhead. The experimental results in Section V show that our detection mechanism provides better detection speed at reasonably low performance cost, both at high and low sampling granularity. Once collected, these runtime profiles of the victim's process under variable system load conditions are used for training and cross validation of the machine learning models in the next phase. Post-training, the classifiers work with the same hardware events at runtime.

2) Training of machine learning models

The hardware events that are used to profile the target cryptosystem at runtime enable the tool to differentiate between attack and no-attack processes. We collected training data on one million samples evenly distributed among representative execution scenarios for the victim's process. For instance, the samples comprise an equal number of training data for attack and no-attack scenarios. Furthermore, each of these scenarios comprises an equal number of training data being collected under no load, average load and full load conditions. We then train the selected machine learning models with these labelled data. Our training data set is un-biased, *i.e.*, it does not favor one execution scenario over the other. For validation purposes, we apply K fold cross validation technique [53] for each individual model using training data to check its accuracy before deploying these models for runtime detection.

3) Classification & Detection

In the third and last phase, trained individual classifiers in the tool utilize runtime data originating from hardware events for classification and detection purpose. Based on training in the second phase, every model classifies the run-time data in two categories: Attack or No-Attack. A majority vote is then taken by the ensemble model on the individual decisions of selected machine learning models to decide whether the cryptosystem is under attack or not. Detection accuracy is based on how well the model is trained, whereas the speed of detection and performance is dependent on how fast we try to collect the samples, *i.e.*, sampling granularity. The rate of miss classifications is the measure of imperfections and inclinations of the predicted results and is defined as False Positives (FPs) and False Negatives (FNs). FPs is the case when a no-attack condition is detected as an attack and leads to loss of performance and loss of confidence. On the other hand, FNs is a condition in which an attack is detected as a no-attack, which is more of a security breach and involves loss of critical information. The results in Section V show how accurately and rapidly different models classify data samples at runtime in our proposed detection mechanism.

B. HARDWARE PERFORMANCE COUNTERS

Hardware performance counters (HPCs) are special purpose hardware registers present in almost all modern processor families. HPCs are basically used to monitor the performance of applications on architectural events (*e.g.*, cache misses/hits, CPU cycles, cache references, *etc.*) while applications are running on underlying hardware. Processors based on Intel's x86 architecture [52] provide access to hundreds of hardware events that can reveal valuable information of the system using HPCs. However, modifiable HPCs are limited in number. Therefore, few events can be monitored concurrently (ranging from 4 to 8 events). There are many high-level libraries and APIs that can be used to configure and read HPCs, such as; PerfMon [54], OProfile [55], Perftool [56], Intel Vtune Analyzer [57] and PAPI [58], *etc.* As mentioned in Section II-B, many detection techniques use HPCs to detect different CSCAs. Selection of most appropriate and minimum number of hardware events that could help revealing the attack behavior remains an important challenge for detection tools.

1) Selection of HPCs

Many hardware events provide valuable information regarding normal vs abnormal behavior of running processes. For instance, Figure 2 shows some experimental results of selected hardware events (under NL conditions) that measure L1-Data Cache Misses (L1-DCM), L3-Total Cache Accesses (L-TCA) and L3-Total Cache Misses (L3-TCM) and Total Cycles for $100,000$ encryptions of the AES cryptosystem. Figure 2 shows the frequency of samples on the Y-axis and the magnitude of measured event on the X-axis. Results shown in green represent normal behavior of AES encryption algorithm running under no-attack, and results in red show AES running under Prime+Probe attack. Figure 2 shows that the magnitude of these events varies particularly (increases in this case) under an attack situation compared to in a no-attack (normal) situation, indicating that the events are particularly affected during the attack. Our detailed experiments with other cache-related hardware events show that they reveal very interesting information about CSCAs.

TABLE 2: Selected events related to CSCAs

Scope of Event	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

Since we target access-driven CSCAs, we consider only hardware events that are the most affected by these attacks. We performed experiments on a larger set of related hardware events (25+), including cache and system-wide counters and selected the 12 most significant events depicting the cache behaviors, as shown in Table 2. These events have also been used in previous studies for CSCA detection [59], [60], [12], [32]. Any of these events can be useful for detecting CSCAs relying on cache behaviors. Selection of events is

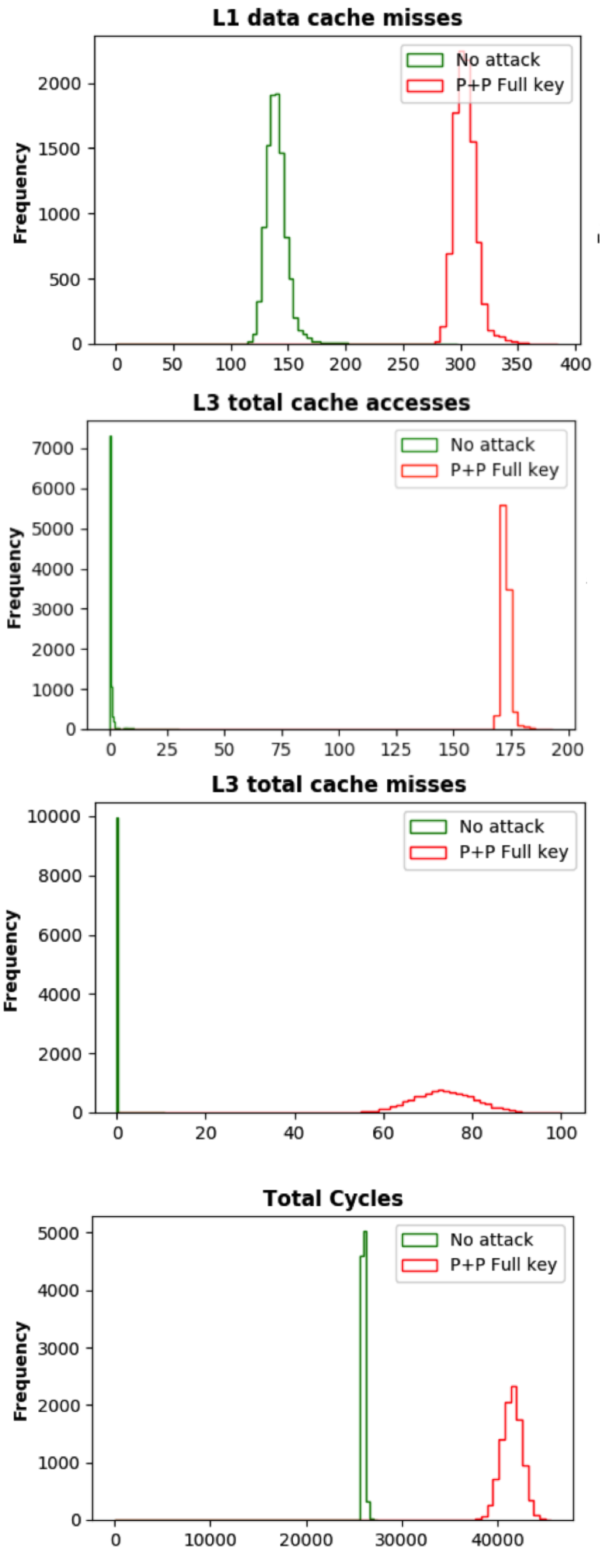

FIGURE 2: Results showing selected hardware events under no load condition for Prime+Probe attack on AES.

TABLE 3: Selected events related to use-case CSCAs

Attack	Hardware Event as Feature	Feature ID
Flush+Reload	L1- Data Cache Misses	L1-DCM
	L3-Total Cache Accesses	L3-TCA
	L3-Total Cache Misses	L3-TCM
	Total CPU Cycles	TOT_CYC
Flush+Flush	L1- Data Cache Misses	L1-DCM
	L3-Total Cache Misses	L3-TCM
	L3-Total Cache Accesses	L3-TCA
	Total CPU Cycles	TOT_CYC
Prime+Probe	L1-Data Cache Misses	L1-DCM
	L3-Total Cache Accesses	L3-TCA
	L3-Total Cache Misses	L3-TCM
	Total CPU Cycles	TOT_CYC

strictly based on the following five arguments: 1) the relevance of events with respect to attack's behavior, 2) provision of precise and distinctive information on normal/abnormal behavior that does not overlap with any other set of events, 3) diversity and non-correlation among features that can be fed to machine learning models to enable confidant decision-making under realistic scenarios, 4) a minimum but sufficient set of events that does not cause huge performance overhead while sampling, and 5) minimum events that do not require multiplexing of hardware performance counters (as multiplexing can lead to non-deterministic and imprecise measurements at run time). Since every selected attack has its own peculiar characteristics and yet they affect caches in one way or the other, for the WHISPER tool, we chose the most suitable minimum number of hardware events that could maximize the understanding of targeted vulnerabilities in the cache behavior. To elaborate further, for instance, almost all cache-based side-channel or covert-channel attacks target the sharing property of the caching hardware (e.g., inclusive LLC) or use specialized instructions like *CLFLUSH* instruction. They increase the cache miss rate by continuously flushing selected cache lines, which also affects the total CPU time for the victim's process. Thus, we select the hardware events that are the most affected ones by the cache's behavior when system is under attack. As illustrated in Figure 2, we can see that attack behavior is mostly influenced by these 4 events; attacker creates a significant rise in L1 data cache misses and L3 total cache misses due to continuous flushing, the total number of access of cache increase and distinguishes normal/abnormal behaviors due to the number of total cycles increase as attacker is performing activity along with victim's execution. It is important to note that tweaking HPCs with relevance to attack behavior can provide a lot of information on system behavior to launch a detection mechanism.

Based on the CSCA characteristics reported in Section III, we selected the hardware events mentioned in Table 3 as being the best suited. For instance, figures 2 and 3 present experimental results of these selected hardware events for Prime+Probe attack under no load and full load conditions,

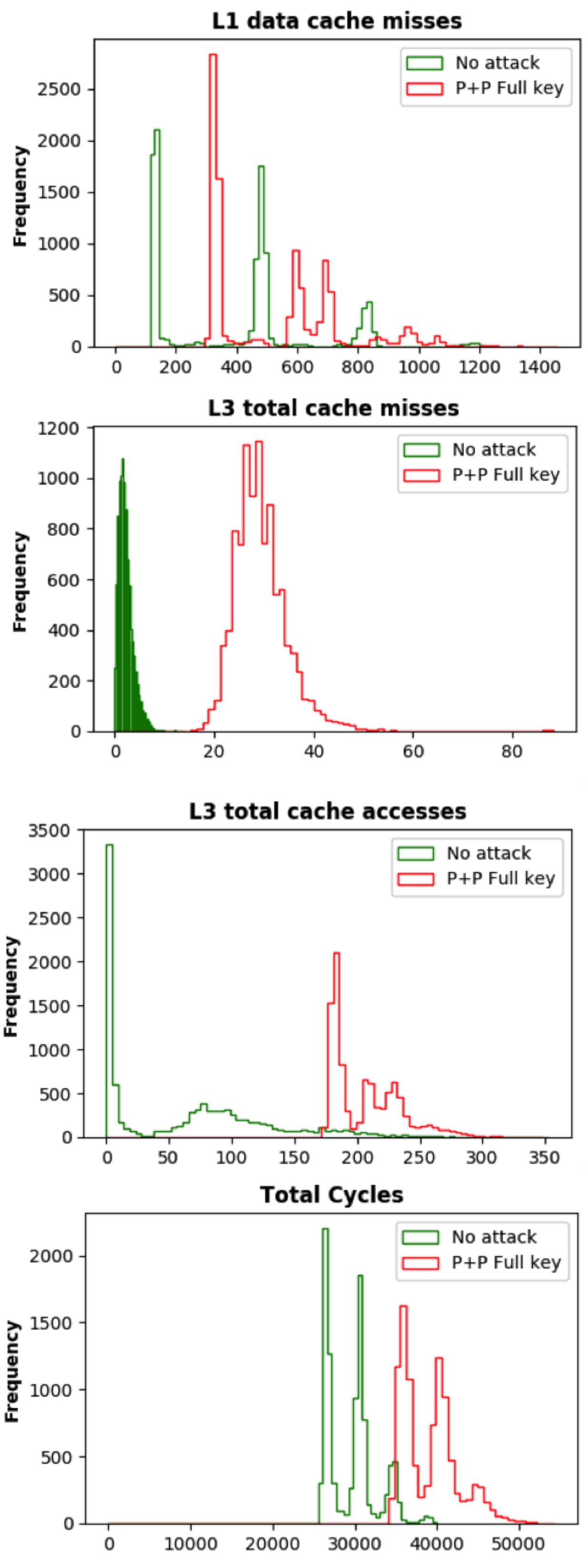

FIGURE 3: Results showing hardware events under Full Load conditions for Prime+Probe attack on AES.

respectively. As depicted in these figures, each hardware event shows distinguishable features for attack and no-attack cases.

TABLE 4: List of Machine Learning Models for CSCA Detection (Non-exhaustive)

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 (DT)	Non-linear
11	Dummy	Non-linear
12	Neural Networks	Non-linear

C. MACHINE LEARNING MODELS

In this section, we discuss the rationale for selecting machine learning models for the WHISPER tool. In section IV-B, we discuss how hardware events can provide valuable information regarding the behavior of target processes and how the load conditions can affect their ability to provide distinguishable information. The difference between attack and no-attack scenarios is quite clear under No Load conditions as shown in Figure 2, which could easily be separated using a threshold. However, under a more realistic load condition such as Full Load, this situation worsens, as shown in Figure 3. Due to increased interference with caches, it becomes hard to separate an attack scenario from a no-attack scenario with simple threshold-based approaches.

Adding to the problem, in practice, a system can be exposed to multiple CSCAs simultaneously or in any temporal order, which would further increase the difficulty in distinguishing an attack scenario using data from hardware events. To explain this, we conducted experiments with six different attacks as use-cases (discussed in Section III). We collected data for selected hardware events under all load conditions for these six use-cases separately and then plotted them together as shown in Figure 4 for the full load case in order to illustrate the overlapping nature of data under different attacks. As can be seen in Figure 4, it is not easy to classify these HPC's data using simple threshold-based approaches.

For such a system, machine learning models can be helpful by appropriately learning the behavior of each CSCA using HPC data. However, any selected ML model is assumed to deal with such a data mix in order to separate an attack from a no-attack scenario, whereas our experiments show that not

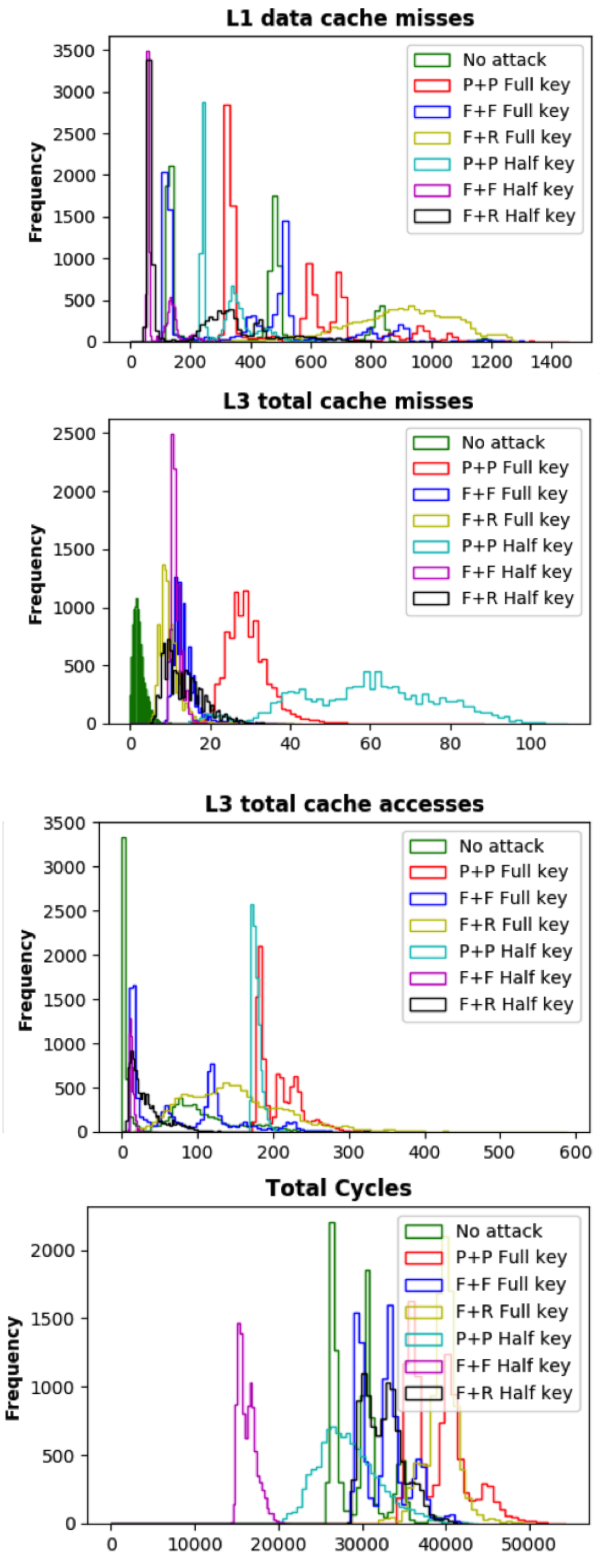

FIGURE 4: Results showing selected hardware events under Full Load conditions for six CSCAs on AES.

all ML models yield acceptable results and it is important to understand the diversity of data affecting the caches in an attack/no-attack scenario for selection of ML models. To this end, in Section IV-C1, we discuss the behavior of mixed data collected under all six attacks that affects the caches. Analysis of these data provides an insight about the data mix and helps select appropriate ML models for the WHISPER tool. Based on the analysis in Section IV-C1, we then discuss the selection of ML models in Section IV-C2.

1) Dimensionality Reduction of Data

Before we present the results for different use-cases, it is important to have a look at the data that are being used for classification. Since the WHISPER tool is designed to detect multiple attacks at runtime, therefore, also for training purposes, the tool simultaneously takes into account the data from six attacks exhibiting different behaviors. It becomes very difficult for the ML models to perform behavioral analysis on such a mix of data. Figure 5 illustrates the data density plot between two selected hardware events, L1_DCM & L3_TCA in this case, under Full Load conditions for combined data of all attacks. These results show that the ML features are heavily correlated, which is the reason why not all linear models were able to perform well on these data. Other density plots between selected hardware events reveal similar information and are therefore omitted in favor of space.

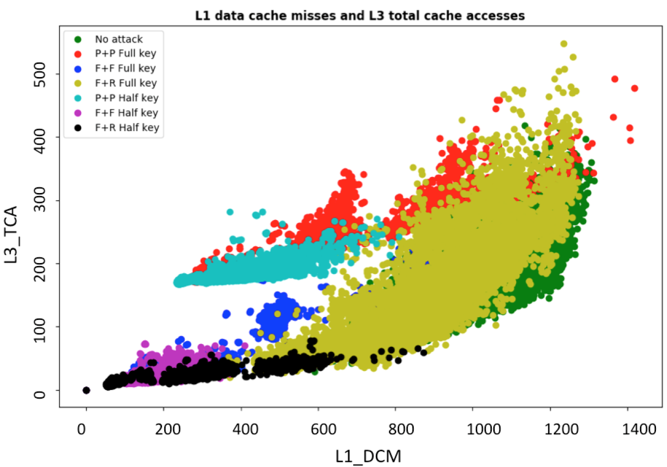

FIGURE 5: Results for data density between L1_DCM & L3_TCA under Full Load conditions -All attacks combined.

For the sake of clarity, we reduced the dimensionality of data for visualization using t-distributed Stochastic Neighbor Embedding (t-SNE) algorithm [63]. Other tools can be used for data reduction including Principal Component Analysis (PCA) [64]. PCA is a mathematical technique but t-SNE is a probabilistic one. In order to represent high dimension data on low dimension, non-linear manifold, similar data points have to be represented close together, which can be more easily achieved by t-SNE rather than with PCA. As a result, in Figure 6, the data reduction can be seen from high scale to a good classification scale for Full Load conditions. This representation of data, on the one hand, provides us with the necessary information to select best-suited ML models for the tool and, on the other hand, provides a good insight into the cache behavior under multiple attacks. Figure 6 shows that, in reduced dimensions, the no-attack case is still distinguishable from the attack case if appropriate ML models are used.

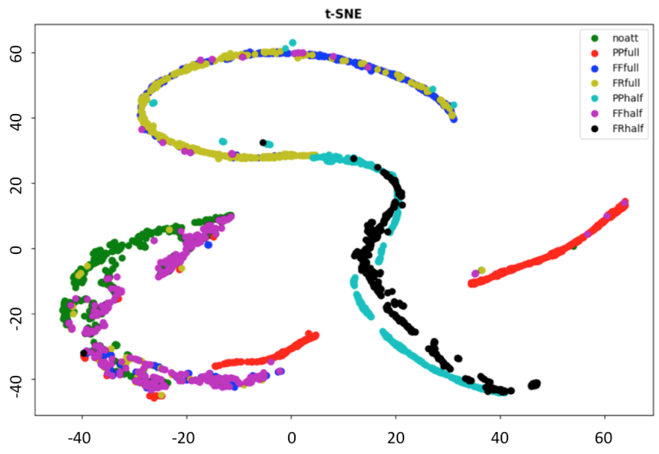

FIGURE 6: Results on data reduction and visualization using t-SNE algorithm under Full Load conditions -All attacks combined.

2) Selection of Machine Learning Models

Machine learning models are broadly classified into two basic categories; linear and non-linear models. We experimented with a set of 12 distinguished ML models, among which, 6 models are linear and 6 models are non-linear as mentioned in Table 4. LDA is a statistical model used in Machine Learning to discriminate classes by finding a linear combination of the input features such that it provides the best separation between classes. SVM is a non-probabilistic supervised Machine Learning model also known as maximum margin classifier. SVM learns and chooses an optimal hyperplane through the training data in such a way that its distance from the nearest training data points is maximum on each side. The hyperplane then characterizes the new input data points. QDA is a statistical, non-linear, supervised model used in Machine learning. QDA is somewhat similar to LDA in the context of assumptions, except for one assumption of the same co-variance matrix between the features. QDA finds a non-linear combination of features such that it provides the best separation between classes. RF is an ensemble learning method that is used for classification and regression purposes. It is mostly used for over-fitting of training sets. It is robust to the inclusion of irrelevant features and produces inspectable models. It explodes in the form of a tree and is able to grow deeply following highly irregular patterns. KNN is a non-parametric statistical approach used in pattern recognition and for supervised classification in Machine Learning. KNN classifies an incoming data point by assigning it the same label as that of its maximum K-nearest training data points label. This algorithm is computation and memory intensive. The Nearest Centroid is a parametric supervised classifier used in ML. It calculates the distance of the new input data point from the mean of the training data of each class and then assigns the label to the new input data point in the class whose calculated distance was smallest. Naive Bayes is a probabilistic supervised ML classifier with a strong assumption that the features are independent of each other. It calculates the probability of an incoming data point in each class. The new data point gets the label of the class whose probability is greater. Perceptron is like the neuron in human brain. It is a linear supervised ML approach which represents the simplest neural network. It learns some weights for future use from the training data and then predicts the class of the new incoming data point by calculating the weighted sum of these features. DT is a tree like model of decisions and their possible consequences. It is one of the models that contains conditional control statements. In decision analysis, DT mainly helps to carve a strategy to reach certain goals. Dummy Model is a naive approach that can be used for classification. It assigns the label of the most frequent class to the new input data. Neural Networks, also known as multilayer perceptrons, are composed of multiple perceptron hidden layers. Feed forward and backward propagation techniques can be used for error reduction. Detailed explanations about these models are available in [53], [61], [62], [63]. We performed experiments with well-known basic ML models as they exhibit low overhead at runtime with considerable accuracy. This list is non-exhaustive and does not imply rejection of the use of other ML models.

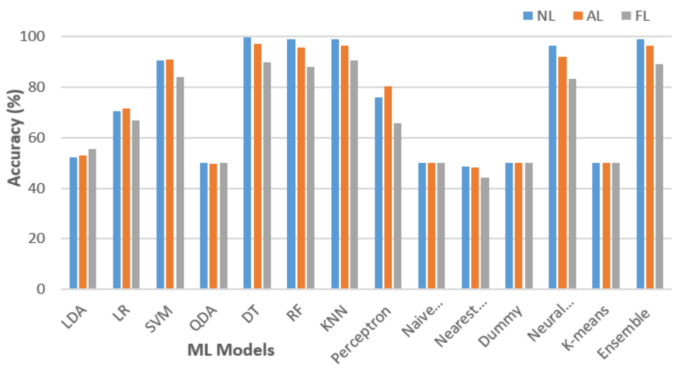

FIGURE 7: Accuracy Comparison of ML models for 6 Attacks

Since WHISPER is a runtime CSCA detection tool, we apply stringent criteria for selection of ML models that best suit our design constraints, *i.e.*, classification accuracy, implementation feasibility for run-time detection, performance overhead, distribution of error (false positives and false negatives) and detection speed. Detection accuracy is the most important criterion for the tool. Therefore, we test the detection accuracy for all 12 models with a training data set collected for all six attacks running on the system. We then compare these models on the rest of the design constraints. Figure 7 illustrates the detection accuracy of all 12 models that were tested against six attacks with variable load conditions (No Load, Average Load and Full Load). Figure 7 shows that linear models, including LDA, LR, Naive Bayes, Nearest Centroid and Perceptron do not perform very well on detection accuracy while classifying the HPC data of six attacks. This is because of the significant data overlap. Similar pattern can also be observed in some of the non-linear models, including QDA, Dummy and K-means. Based on the detection accuracy alone, it leaves us with a choice of SVM, DT, RF, KNN and Neural Networks. However, detection accuracy, the most important parameter, is not the only one to consider when deploying a high-speed run-time CSCA detection tool. The other very important parameter to examine while comparing ML models is their implementation feasibility. Likewise, ML models should be able to quickly provide their decision while keeping their performance overhead minimum. According to these criteria, although the KNN model shows good detection accuracy, it uses all training data points at runtime to infer decision. Thus, KNN involves significant implementation complexity, which can lead to high performance and storage overheads. Neural network is a model that performs well in terms of detection accuracy as shown in Figure 7. It uses a forward and backward propagation technique that causes the problem of over-fitting due to which it backtracks all events for error reduction and eventually increases performance overhead. That is why we did not use neural networks in the WHISPER tool, but it could be a useful selection for sophisticated attacks in future. For the moment, our problem can be resolved using models that provide less overhead than Neural Networks. That is why we selected the remaining three models with a good performance, *i.e.*, DT, RF and SVM. All these models perform consistently, offer good accuracy and ease of implementation while providing negligible performance overhead (as demonstrated by experimental evidence). The WHISPER tool uses these three ML models to demonstrate its effectiveness. However, the tool is scalable and can include/exclude other ML models with one-time training phase, as explained in Section IV-A.

Our experiments revealed that, although all three selected ML models perform reasonably well when used individually, they still exhibit high rate of false positives and false negatives because of the variation in HPC data captured under different attacks. Therefore, the WHISPER tool uses all three models in an ensemble fashion where the final decision on detection is taken through a majority-vote among the best-performing ML models for runtime input data. The results in Figure 7 show that the ensemble of models shows consistent accuracy for diverse data coming from six different attacks at runtime.

D. IMPLEMENTATION OF DETECTION MODULE

Before presenting the experimental evaluation, in this section, we briefly present the implementation details of the WHISPER tool. Algorithm 1 presents an abstract level pseudocode for WHISPER's detection module. As illustrated, the module takes as input the sampling granularity for hardware events (`SamplingGranularity`), which can be either user-defined or adjusted at runtime. By default, the sampling granularity is set to fine grain, *i.e.*, 10 AES encryptions. The sampling in WHISPER is user-defined and can be changed based on the type and the level of threat at runtime. Moreover, different attacks require observation of a different number of encryptions in order to be successful. For instance, there are certain attacks that require observation of only a single encryption to complete (e.g., Flush+Reload on RSA-1024 bits [3]) while other attacks require observtion multiple encryptions to complete (*e.g.*, Flush+Flush on AES [4]). Therefore, in the case of Flush+Reload on RSA, WHISPER can set the sampling granularity at the rate of every 10 bits, whereas 10 encryptions for Flush+Flush on AES require observation of roughly $350 - 400$ encryptions to complete the attack. Sampling granularity can always be tuned by the user/system to effectively control the performance overhead depending on the type and level of threat at runtime. Another input is the total number of iterations for which we tested the module (`MaxIterations`). The number of iterations varies for each attack as discussed in Section V. Lines $1 - 3$ show that a victim's (encryption) process is initialized, the detection module is activated once and the hardware events are set around the victim's process considering it as the ROI (Region of Interest). For the overall selected number of iterations, the module activates detection after every 10 or 100 encryptions in the case of AES cryptosystem, depending on what granularity has been specified through `SamplingGranularity` (lines $4-6$). Once activated, the detection module collects the data from hardware events (line 7) and feeds them as features to selected binary classifiers (line 8) in order to perform individual voting. Based on these individual votes, the module generates a report after taking a majority vote on the results (line 9). Detection is then deactivated (line 10) and if the report is `True` then an attack is reported (lines $11 - 12$). Otherwise, the victim's process continues to execute uninterrupted.

Algorithm 1: Run-time Detection Module

```
   Input: SamplingGranularity, MaxIterations
   Initialization:
   events← ∅, votes ← ∅
   report← False, VictimProcess ← NIL
1  VictimProcess← Get_Encryption_Process()
2  Activate_Detection(VictimProcess)
3  Set_Hardware_Events(VictimProcess)
4  for i ← 1 to MaxIterations 1 do
5      if i mod SamplingGranularity == 0 then
6          Activate_Detection()
7          events ← Read_Hardware_Events()
8          votes ← ML_Classifiers(events)
9          report ← Majority_Voting(votes)
10         Sleep_Detection()
11         if report == True then
               /* Attack Detected              */
12             return 1
   /* No-attack detected !                  */
13 return 0
```

V. EXPERIMENTS AND DISCUSSION

We evaluate the WHISPER tool under stringent design constraints. To do so, we create three experimental case studies to detect Prime+Probe, Flush+Reload and Flush+Flush attacks, respectively, on AES cryptosystem. In each case, we evaluate the performance of our tool under variable load conditions, *i.e.*, No Load (NL), Average Load (AL) and Full Load (FL) conditions, as explained in Section I.

A. SYSTEM MODEL

We demonstrate the efficiency of the WHISPER tool on Intel's core $i7-4770$ CPU running on Linux Ubuntu 16.04.1 with Kernel $4.13.0 - 37$ at 3.40-GHz. Our threat model consists of access-driven CSCAs, which produce information leakage over the entire cache hierarchy (L1, L2 & LLC) of Intel x86 architecture. We take same-core and cross-core CSCAs into account for the purpose of detection. We consider that the Operating System (OS) is not compromised in our system model (*i.e.*, it is part of the trusted computing base). The PAPI (Performance Application Programming Interface) library is used to access hardware events using HPCs from the Intel Core $i7$ machine. It provides machine and operating system independent access to HPCs. Any of over 100 preset events can be counted through either a simple high level programming interface or a more complete low level interface of PAPI. For experimentation, the training has been performed offline with 1-Million samples of unbiased data of attack and no-attack samples. For run-time inference, we analyzed the results by running the attacks 100,000 times. In the following, we present our experimental results using 3 case studies. Additionally, for comparative analysis, we provide results using individual ML models running separately as well as in an ensemble fashion.

B. CASE STUDY-I: DETECTING PRIME+PROBE

Our first case study provides experimental evaluation of the detection of two implementations of a Prime+Probe attack targeting the AES cryptosystem.

1) Detection Accuracy

Detection accuracy is the primary indicator to judge the effectiveness of any detection tool. In all our case studies, we use percentage accuracy to demonstrate the validity of trained machine learning models. We use unbiased samples in the training and validation data, *i.e.*, samples for attack and no-attack cases are equal.

Tables 5 & 6 show our experimental results for individual ML models (RF, DT, SVM) and Ensemble, respectively, for two different implementations of the Prime+Probe attack. These results illustrate the variation in aforementioned metrics under different load conditions. All three ML models individually provide very high and consistent detection accuracy under No Load, Average Load and Full Load conditions, *i.e.*, between 92.67–99.99% for Impl.1 (Table 5) and between 97.73–99.99% for Impl.2 (Table 6). Evidently, the Ensemble model used by the WHISPER tool also performs very well and provides a detection accuracy ranging between 97.62–99.99% for Impl.1 (Table 5) and 96.94–99.77% for Impl.2 (Table 6), respectively. The results of Ensemble model are particularly interesting under Average Load and Full Load conditions where individual models might not always perform consistently. To support these results, we also provide the run-time behavior of hardware events under different load conditions. For instance, Figure 8 shows the behavior of hardware events for Prime+Probe attack Impl.1 under Full Load conditions and Figure 2 shows the same events for Prime+Probe attack Impl.2 under No Load condition. Figures 2 & 8 illustrate that, under Prime+Probe attack, the hardware events offer distinguishable behavior for attack and no-attack scenarios under all load conditions, which is why the detection accuracy for all ML models remains very high.

2) Detection Speed

Detection speed is another important criterion for the evaluation of any run-time intrusion detection tool. Detection speed is an indirect reflection of how aggressively a detection tool profiles the victim's process behavior (through hardware events in this case) and provides its decision. Detection speed also affects the resulting performance overhead of the tool as it is a trade-off between how fast an intrusion can be detected versus how much overhead the detection process would cost. According to the literature, a Prime+Probe attack would require AES cryptosystem to perform at least $4,800$ encryptions for Impl.1 and $50,000$ encryptions for Impl.2 [4], [6] in order to be successful. Therefore, the percentage of encryptions being performed before the WHISPER tool raises a flag, defines the detection speed with respect to attack completion. For instance, for Prime+Probe attack Impl.1, if the tool raises a flag before 4800 encryptions of AES are performed, then the tool is said to be capable of detecting a Prime+Probe attack on AES within 10% of attack completion. Please note that the detection speed is determined in a time-independent manner. Also, theoretically, it is considered enough for an attacker to deduce 50% of the secret key, as the rest of the key can be acquired using reverse engineering techniques [2], [5]. Therefore, it is safe to detect an attack before it can complete at most 50% by itself. For the WHISPER tool, we considered this the upper bound on detection speed.

Our experimental results in Tables 5 & 6 show that the WHISPER tool is capable of detecting Prime+Probe attack Impl.1 and Impl.2 within 0.21% and 0.02% of completion, respectively. This result implies that the WHISPER tool detects within the first 10 encryptions out of 4800 and $50,000$ encryptions required by Impl.1 and Impl.2, respectively. This speed is achieved under variable load conditions with fine-grain sampling frequency. Fine-grain is the highest profiling granularity used in these experiments in which the tool samples hardware events after every 10 encryptions. We also tested the tool with coarse-grain sampling granularity, as shown in Tables 7 & 8. Coarse-grain profiling would mean sampling hardware events after every 100 encryptions. Tables 7 & 8 show that the resulting accuracy remains almost the same or is further improved in some cases, while the system overhead decreases drastically compared to fine-grain sampling.

Results in Tables 7 & 8 reveal that the tool is still capable of achieving detection accuracy comparable to that of fine-grain detection, while the performance overhead is reduced by significantly large margins. For instance, in Tables 6 & 8

TABLE 5: Results using individual and Ensemble ML models for detection of Prime+Probe (*Impl.1*: half-key recovery) on AES at fine-grain sampling.

ML Model	Load	Accuracy (%)	Speed (%)	FP (%)	FN (%)	Overhead (%)
DT	**NL**	99.99	0.21	0.01	0.00	6.59
	AL	99.76	0.21	0.24	0.00	
	FL	95.00	0.21	5.00	0.00	
SVM	**NL**	99.99	0.21	0.01	0.00	7.83
	AL	99.82	0.21	0.18	0.00	
	FL	94.92	0.21	5.08	0.00	
RF	**NL**	99.99	0.21	0.01	0.00	11.3
	AL	99.72	0.21	0.28	0.00	
	FL	92.67	0.21	7.33	0.00	
Ensemble	**NL**	99.99	0.21	0.01	0.00	8.03
	AL	99.77	0.21	0.23	0.00	
	FL	97.62	0.21	2.37	0.01	

TABLE 6: Results using individual and Ensemble ML models for detection of Prime+Probe (*Impl.2*: full-key recovery) on AES at fine-grain sampling.

ML Model	Load	Accuracy (%)	Speed (%)	FP (%)	FN (%)	Overhead (%)
DT	**NL**	99.99	0.02	0.01	0.00	7.45
	AL	98.53	0.02	1.47	0.00	
	FL	98.54	0.02	1.46	0.00	
SVM	**NL**	99.99	0.02	0.01	0.00	6.75
	AL	98.50	0.02	1.50	0.00	
	FL	98.61	0.02	1.38	0.02	
RF	**NL**	99.99	0.02	0.01	0.00	9.34
	AL	97.90	0.02	2.10	0.00	
	FL	97.73	0.02	2.27	0.00	
Ensemble	**NL**	99.77	0.02	0.23	0.00	8.20
	AL	96.94	0.02	3.6	0.00	
	FL	99.09	0.02	0.91	0.00	

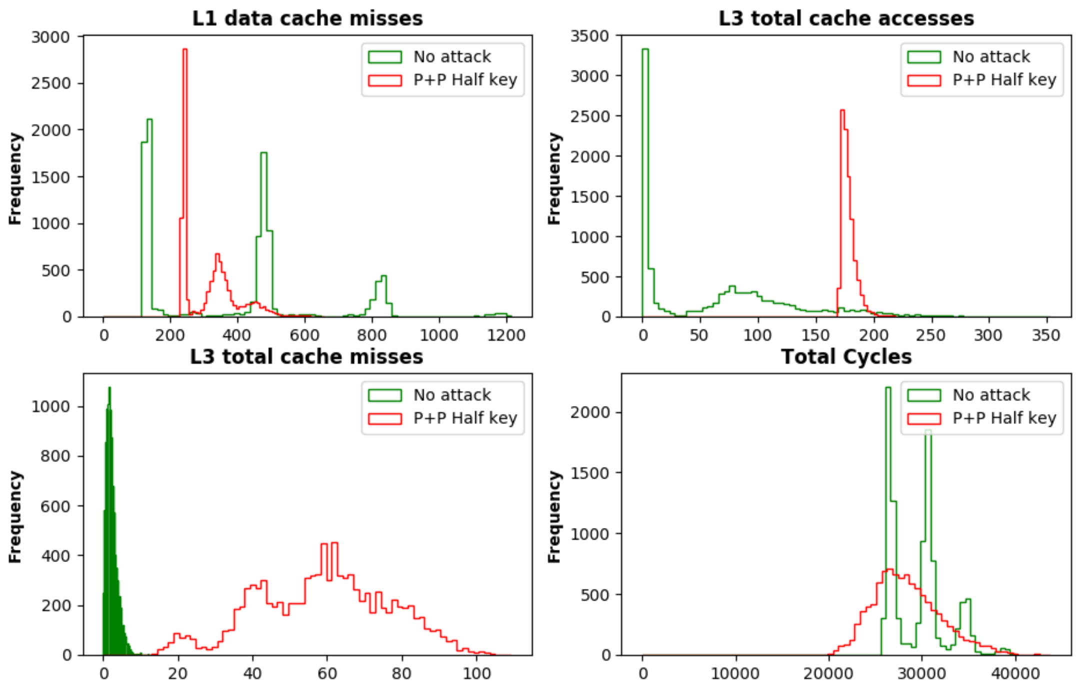

FIGURE 8: Run-time behavior of selected hardware events under Full Load conditions for Prime+Probe *Impl.1*

with Prime+Probe attack Impl.2, the performance overhead for the Ensemble model decreased from 8.20% to 1.03% –that is, by a factor of roughly 8! Similarly, the misclassification rate for FPs and FNs also decreases due to coarse-grain detection where, in most cases, FNs are always zero or negligible. Detection speed would naturally go down by a small margin in the case of coarse-grain detection as we sample hardware events only after every 100 encryptions.

TABLE 7: Results using individual and Ensemble ML models for detection of Prime+Probe (*Impl.1*: half-key recovery) on AES at coarse-grain sampling.

ML Model	Load	Accuracy (%)	Speed (%)	FP (%)	FN (%)	Overhead (%)
DT	**NL**	99.99	2.08	0.01	0.00	0.30
	AL	99.82	2.08	1.47	0.18	
	FL	98.18	2.08	1.82	0.00	
SVM	**NL**	100	2.08	0.00	0.00	0.15
	AL	97.51	2.08	2.49	0.00	
	FL	98.72	2.08	1.28	0.00	
RF	**NL**	99.99	2.08	0.01	0.00	2.99
	AL	97.20	2.08	2.80	0.00	
	FL	97.70	2.08	2.30	0.00	
Ensemble	**NL**	99.99	2.08	0.01	0.00	3.01
	AL	97.48	2.08	2.52	0.00	
	FL	97.93	2.08	2.07	0.01	

TABLE 8: Results using individual and Ensemble ML models for detection of Prime+Probe (*Impl.2*: full-key recovery) on AES at coarse-grain sampling.

ML Model	Load	Accuracy (%)	Speed (%)	FP (%)	FN (%)	Overhead (%)
DT	**NL**	99.99	0.19	0.01	0.00	4.5
	AL	99.71	0.19	0.29	0.00	
	FL	98.19	0.19	1.81	0.00	
SVM	**NL**	99.99	0.19	0.00	0.01	0.82
	AL	99.32	0.19	0.66	0.01	
	FL	98.87	0.19	1.13	0.00	
RF	**NL**	99.99	0.19	0.01	0.00	1.71
	AL	99.21	0.19	0.79	0.00	
	FL	97.90	0.19	2.10	0.00	
Ensemble	**NL**	99.99	0.19	0.01	0.00	4.55
	AL	99.29	0.19	2.52	0.71	
	FL	98.19	0.19	1.81	0.00	

3) Confusion Matrix

Confusion matrix provides a prognosis of results by representing the number of correct and incorrect predictions by the ML models as False Positives (FP) and False Negatives (FN). Ideally, any error in the detection is not desired. The presence of FNs in the system are indicative of a direct security breach, whereas, the presence of FPs would generate a false alarm, thus leading to performance degradation in the system. From the point of view of security, having a bounded amount of FPs compared to FNs is still less damaging to the system in the sense that FNs would be a breach of security while FPs would cause degraded performance. For comparative analysis, we look into the FPs and FNs generated by each individual model compared to the Ensemble model. As shown in Tables 5 & 7, for Prime+Probe attack Impl.1, all models offer very high accuracy, which naturally leads to a very low number of FPs & FNs. For instance, if we analyze the miss-classifications performed by these models under variable load conditions, the SVM model misclassifies 5.08% for fine-grain detection and 2.49% for coarse-grain detection, the DT model misclassifies 5.0% for fine-grain detection and 1.82% for coarse-grain detection and RF model misclassifies between 7.33% for fine-grain detection and 2.80% for coarse-grain detection, respectively. These are the maximum misclassification results. Compared to all these individual models, the Ensemble misclassifies 2.37% at fine-grain detection and 2.52% at coarse-grain detection in the worst case. A very important observation here is that, while the overall error in classification is already very low, misclassification only concerns False Positives. No False Negatives are generated by the tool, except for 0.01% of FN by the Ensemble model under FL conditions for fine-grain detection, which is negligible. For coarse-grain detection, the misclassification rate is almost always 0% for FP & FN.

We report similar results for Prime+Probe attack Impl.2 as shown in Tables 6 & 8. Here, SVM model miss-classifies between 1.50% for fine-grain detection and 1.13% for coarse-grain detection, DT model misclassifies between 1.47% for fine-grain detection and 1.81% at coarse-grain detection and RF model misclassifies between 2.27% for fine-grain detection and 2.10% for coarse-grain detection, respectively, in the worst-case. Compared to all these individual models, the Ensemble misclassifies between 3.6% for fine-grain detection and 2.52% for coarse-grain detection in the worst-case. For Prime+Probe attack Impl.1, the Ensemble model performs well for fine-grain detection and further reduces the misclassification rate for coarse-grain detection. For Impl.2, at coarse-grain detection, there are only 0.71% FNs in Average Load condition but in all other cases it implies 0.00% FNs.

4) Performance Overhead

The performance overhead of the detection tools becomes a particularly important design parameter in the case of runtime detection. Moreover, the adaptability and scalability of the tool also depends on its runtime performance overhead. As discussed briefly in Section V-B2, the detection granularity determines how fast the hardware events are profiled by a detection tool to make effective decisions at run-time. This granularity aggressively impacts the performance overhead as fine granularity would imply more time spent in profiling. We measure performance overhead as a percentage of slowdown experienced by the AES cryptosystem when detection is being enabled. Our experiments with selected ML models reveal that the performance overhead of the WHISPER tool is low, specifically at coarse-grained detection. Tables 5 & 6 show that the AES cryptosystem experiences a maximum slowdown of 11.3% and 9.3% for Impl.1 and Impl.2, respectively, at fine-grain detection granularity for the RF model. A coarse-grain sampling frequency of 100 encryptions for hardware events reduces the performance overhead to a maximum of 2.99% and 4.5% for Impl.1 and Impl.2, respectively, as shown in Tables 7 & 8.

C. CASE STUDY-II: DETECTING FLUSH+RELOAD

Our second case study concerns experimental evaluation of the detection of two implementations of Flush+Reload attack targeting AES cryptosystem.

1) Detection Accuracy

Although a Flush+Reload attack is considered a high-resolution CSCA, the WHISPER tool also demonstrates very high detection accuracy for this attack case study. Tables 9 & 10 show the experimental results of our individual ML models (RF, DT, SVM) and Ensemble, respectively, for two different implementations of Flush+Reload attack. Like the results of our first case study against Prime+Probe attack, all three ML models individually provide very high and consistent detection accuracy under No Load, Average Load and Full Load conditions, *i.e.*, between 97.17–100.00% for Impl.1 (Table 9) and between 98.52–99.99% for Impl.2 (Table 10). Similarly, the Ensemble model used by the WHISPER tool also performs very well and provides a detection accuracy ranging between 98.92–99.99% for Impl.1 (Table 9) and 98.37–99.99% for Impl.2 (Table 10). As shown in Tables 9 & 10, the selected ML models for the tool perform consistently even under Average Load and Full Load conditions against Flush+Reload attack.

The runtime behavior of hardware events gives more insight into these results. For instance, Figure 9 shows the behavior of hardware events for Flush+Reload attack Impl.1 under the No Load condition and Figure 10 shows the same events for Flush+Reload attack Impl.2 under the Full Load condition. These figures show that, under a Flush+Reload attack, the hardware events offer distinguishable behavior for attack and no-attack scenarios.

2) Detection Speed

Flush+Reload attack of the AES cryptosystem requires 250 encryptions for Impl.1 and $50,000$ encryptions for Impl.2 to successfully extract the secret key. Our experimental results, shown in Tables 9 & 10, show that the WHISPER tool is capable of detecting Flush+Reload attack Impl.1 and Impl.2 within 4.00% and 0.02% of their completion, respectively, *i.e.*, within the first 10 encryptions out of 250 and $50,000$ required by the Flush+Reload attack Impl.1 and Impl.2, respectively. Tables 11 & 12 show the results for coarse-grain sampling, where the detection accuracy remains more or less the same, while detection speed is reduced to 40% and 0.2% for Impl.1 and Impl.2, respectively. As illustrated in Tables Tables 11 & 12, the main advantage of using coarse-grain detection remains the significant decrease in

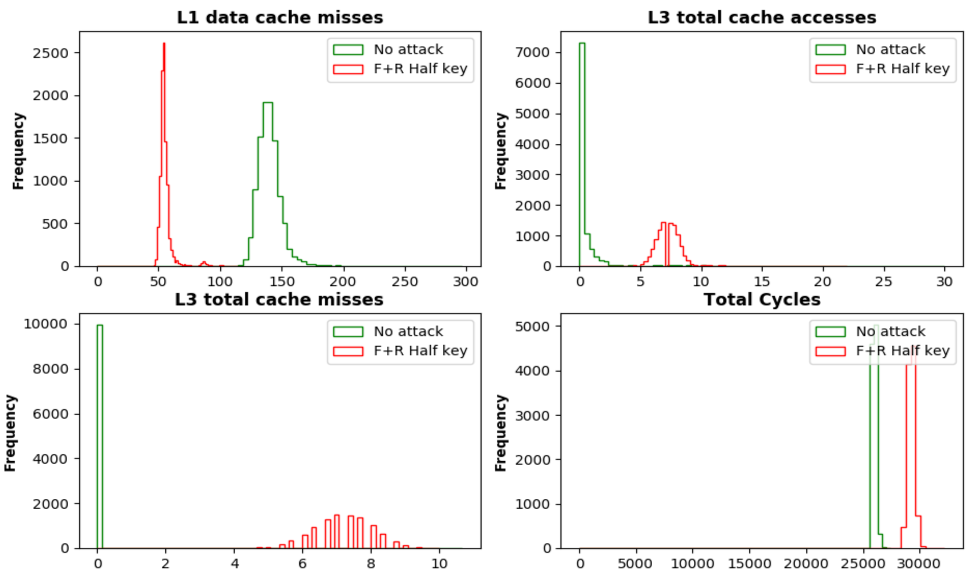

FIGURE 9: Run-time behavior of selected hardware events under No Load conditions for Flush+Reload *Impl.1*

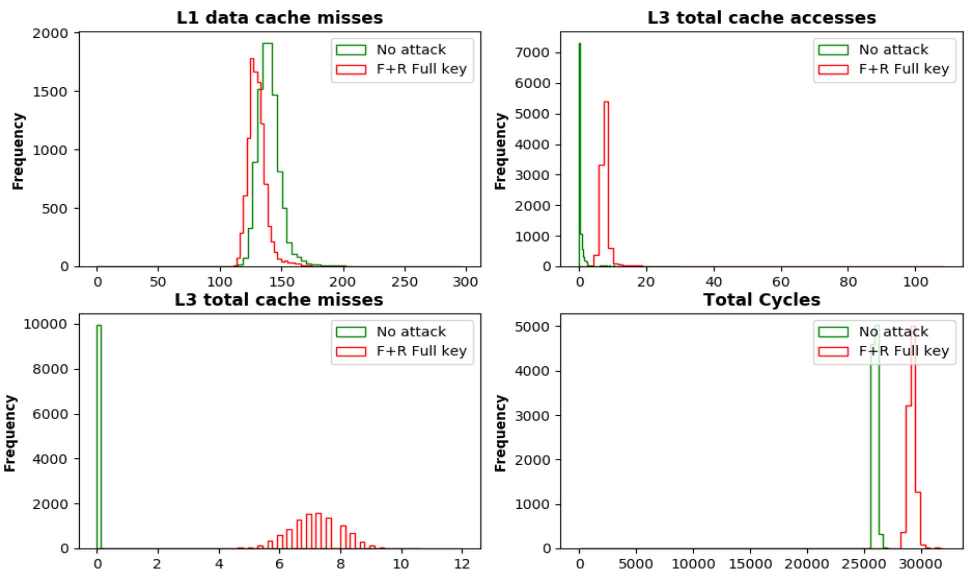

FIGURE 10: Run-time behavior of selected hardware events under Full Load conditions for Flush+Reload *Impl.2*

system overhead compared to fine-grain sampling. Similarly, the misclassification rate in terms of FPs and FNs is also reduced in many cases due to coarse-grain detection. The results show that the percentage of FNs is almost always zero or negligible. This explains why the WHISPER tool works efficiently for coarse-grain detection with a further decrease in system overhead and reasonable detection speed.

One important point to note here and in most of the upcoming results is the constant detection speed. We tested individual as well as Ensemble models with fixed sampling (fine-grain or coarse-grain). Therefore, the attack is detected at a constant speed for both individual and Ensemble models. Moreover, in some cases, the attack detection speed varies slightly due to variable load conditions, but the difference is negligible.

3) Confusion Matrix

We analyzed the misclassification error rate for the WHISPER tool w.r.t. Flush+Reload attack on the same pattern as we did for the Prime+Probe attack. In the Flush+Reload attack Impl.1 (with fine-grain and coarse-grain detection), as shown in Tables 9 & 11, the SVM model misclassifies between 0.86% and 1.53%, DT model misclassifies between 1.0% and 1.77% and RF model misclassifies between 2.83% and 2.17% for fine-grain and coarse-grain

TABLE 9: Results using individual and Ensemble ML models for detection of Flush+Reload (*Impl.1*: half-key recovery) on AES at fine-grain sampling.

ML Model	Load	Accuracy (%)	Speed (%)	FP (%)	FN (%)	Overhead (%)
DT	**NL**	99.99	4.00	0.01	0.00	10.8
	AL	99.71	4.00	0.29	0.00	
	FL	99.00	4.00	1.00	0.01	
SVM	**NL**	100	4.00	0.00	0.00	11.1
	AL	99.75	4.00	0.25	0.00	
	FL	99.14	4.00	0.86	0.00	
RF	**NL**	99.98	4.00	0.02	0.00	12.3
	AL	99.57	4.00	0.43	0.00	
	FL	97.17	4.00	2.83	0.00	
Ensemble	**NL**	99.99	4.00	0.01	0.00	11.2
	AL	99.68	4.00	0.32	0.00	
	FL	98.92	4.00	1.08	0.00	

TABLE 10: Results using individual and Ensemble ML models for detection of Flush+Reload (*Impl.2*: full-key recovery) on AES at fine-grain sampling.

ML Model	Load	Accuracy (%)	Speed (%)	FP (%)	FN (%)	Overhead (%)
DT	**NL**	99.99	0.02	0.01	0.00	11.17
	AL	99.44	0.02	0.56	0.00	
	FL	98.91	0.02	1.09	0.00	
SVM	**NL**	99.99	0.02	0.01	0.00	9.78
	AL	99.45	0.02	0.50	0.05	
	FL	95.92	0.02	0.70	3.38	
RF	**NL**	99.99	0.02	0.01	0.00	13.25
	AL	99.05	0.02	0.95	0.00	
	FL	98.52	0.02	1.47	0.01	
Ensemble	**NL**	99.99	0.02	0.01	0.00	8.27
	AL	98.37	0.02	0.63	0.00	
	FL	98.99	0.02	1.01	0.01	

TABLE 11: Results using individual and Ensemble ML models for detection of Flush+Reload (*Impl.1*: half-key recovery) on AES at coarse-grain sampling.

ML Model	Load	Accuracy (%)	Speed (%)	FP (%)	FN (%)	Overhead (%)
DT	**NL**	99.99	40	0.01	0.00	2.00
	AL	99.86	40	0.13	0.02	
	FL	98.05	40	1.77	0.18	
SVM	**NL**	100	40	0.00	0.00	1.3
	AL	98.47	40	1.53	0.00	
	FL	98.82	40	1.18	0.00	
RF	**NL**	99.99	40	0.01	0.00	1.89
	AL	98.28	40	1.72	0.00	
	FL	97.83	40	2.17	0.00	
Ensemble	**NL**	99.99	40	0.01	0.00	2.19
	AL	98.44	40	1.56	0.00	
	FL	98.09	40	1.91	0.00	

TABLE 12: Results using individual and Ensemble ML models for detection of Flush+Reload (*Impl.2*: full-key recovery) on AES at coarse-grain sampling.

ML Model	Load	Accuracy (%)	Speed (%)	FP (%)	FN (%)	Overhead (%)
DT	**NL**	99.96	0.2	0.02	0.02	0.1
	AL	95.83	0.2	0.36	3.81	
	FL	98.30	0.2	1.70	0.00	
SVM	**NL**	100	0.2	0.00	0.00	0.41
	AL	98.44	0.2	1.56	0.00	
	FL	90.00	0.2	0.99	0.00	
RF	**NL**	100	0.2	0.00	0.00	2.0
	AL	98.20	0.2	1.80	0.00	
	FL	98.41	0.2	1.59	0.00	
Ensemble	**NL**	100	0.2	0.00	0.00	2.10
	AL	98.37	0.2	1.63	0.00	
	FL	98.60	0.2	1.40	0.00	

detection, respectively, in the worst case. Compared to all these individual models, the Ensemble model misclassifies between 1.08% and 1.01% of fine-grain and coarse-grain detection, respectively, in the worst case. The overall error in classification is also very low in this case and the misclassification mainly constitutes False Positives. It can be seen in Table 10 that Ensemble achieves 0.01% of FNs even in cases where individual models do not exhibit any FNs. This is because individual models, compared to Ensemble, are average results of different iterations and sometimes vary due to different runtime conditions. No False Negative is generated by the tool except in the DT model that provides 0.01% in Table 9 and $0.02\% - -0.18\%$ in Table 11. Similar results for Flush+Reload attack Impl.2 are shown in Tables 10 & 12. Here,the SVM model misclassifies between 0.70% and 1.56%, the DT model misclassifies between 1.09% and 1.70% and the RF model misclassifies between 1.47% and 1.80% only in fine-grain and coarse-grain detection, respectively, in the worst case under variable load conditions. The Ensemble model misclassifies between 1.01% and 1.63% in the worst case. The Ensemble model provides only 0.01% FNs under Full Load conditions, as shown in Table 10 and provides no FN, as shown in Table 12 for Flush+Reload Impl.2. If we compare all the results of misclassifications from No Load to Full Load conditions, it can be observed that coarse-grain detection helped reduce FPs and FNs in many cases. This is due to the fact that HPCs are inherently imprecise and non-deterministic, as discussed in Section IV-B1, which might lead to erroneous results in the case of fine-grain (rapid) sampling. A coarse-grain (slow) sampling would allow hardware events to mature their measured values before being read.

4) Performance Overhead

As discussed earlier, performance overhead is more related to the implementation and sampling granularity of hardware events. Tables 9 & 10 show that the AES cryptosystem experiences a maximum slowdown of 12.3% and 13.25% while detecting Impl1. and Impl2. of Flush+Reload attack, respectively, under fine-grain detection granularity. This overhead is remarkably reduced to a maximum of 2% for both implementations of Flush+Reload attack in the case of coarse-grain detection.

D. CASE STUDY-III: DETECTING FLUSH+FLUSH

Our third and last case study provides experimental evaluation of the detection of two implementations of Flush+Flush attack targeting AES cryptosystem. Flush+Flush is a stealth and high-resolution attack that targets LLC except that it is stealthier than Flush+Reload and Prime+Probe attacks. Unlike other CSCAs, the stealth nature of Flush+Flush does not make any memory accesses. Thus, as a running process itself, it causes no cache misses and the number of cache hits are reduced to minimum due to constant cache flushing. Authors in [4] claimed to detect their own attack using hardware events in Linux $Perf\_event\_open$ interface. They documented that the attack (spy process) is non-detectable using 24 hardware events available with Linux syscall interface. Contrary to the claim in the above mentioned paper, we observed that the spy process might not be detectable on one hand, but that, on the other hand, the victim's process is affected by the constant high speed flushing. We build our argument around the fact that detection mechanism should indicate the presence or absence of intrusion from the victim's perspective. Specifically identifying the malicious process is often not required to apply protection. If, and when, an intrusion is detected in the victim's process, the OS can still take preventive measures like complete isolation or execute a critical section of the victim's process, etc. This work practically demonstrates that along with training of other CSCAs, WHISPER can detect stealth nature attacks like Flush+Flush. We achieved considerably high detection accuracy for Flush+Flush attack.

1) Detection Accuracy

In this case study, we demonstrate that Flush+Flush is not only detectable but also detectable with relatively very high detection accuracy using the WHISPER tool. Tables 13 & 14 show our experimental results for individual ML models (RF, DT, SVM) and Ensemble, respectively, for two different Flush+Flush attacks on similar patterns as shown in the other two case studies. In this case as well, all three ML models provide high and consistent detection accuracy when used as individual models under No Load, Average Load and Full Load conditions. As shown, the accuracy ranges between 71.84–99.98% for Impl.1 (Table 13) and between 72.86–99.56% for Impl.2 (Table 14). The Ensemble model performs better than individual ML models in this case and provides a very good detection accuracy ranging between 94.68–99.97% for Impl.1 (Table 13) and 94.82–97.71% for Impl.2 (Table 14). Please note that, in this case study, the use of the Ensemble model instead of individual models has a clear advantage. This is due to the stealthy nature of the Flush+Flush attack, which is not easily detectable by individual models under all load conditions. Thus, a major vote helps achieve the best possible accuracy. As shown in Tables 13 & 14, individual ML models show significant variations under load conditions against Flush+Flush attacks but the Ensemble model performs consistently even under Average Load and Full Load conditions. It can be seen that detection accuracy under Full Load conditions reaches 94% in both cases due to integrating three models in which one model provides poor accuracy, whereas, the other two models are between $94-97\%$ and $95-99\%$. This shows that the Flush+Flush attack is stealthy in nature and hard to detect and we rely on the Ensemble model rather than choosing individual models for detection. For instance, in this case, the SVM predicts with low accuracy. The run-time behavior of hardware events gives more insight for these results. For instance, Figure 13 shows the behavior of hardware events for Flush+Flush attack Impl.1 under No Load condition and Figure 12 shows the same events for Flush+Flush attack Impl.2 under Full Load condition. These figures show that, under a Flush+Flush attack, the hardware events offer much less distinguishable behavior for attack and no-attack scenarios compared to the other two use-cases, which causes the accuracy of ML models to vary. Nevertheless, our tool demonstrates that it is capable of precisely capturing this variation and provides high detection accuracy for stealthier attacks as well.

TABLE 13: Results using individual and Ensemble ML models for detection of Flush+Flush (*Impl.1*: half-key recovery) on AES at fine-grain sampling.

ML Model	Load	Accuracy (%)	Speed (%)	FP (%)	FN (%)	Overhead (%)
DT	**NL**	99.98	2.50	0.01	0.01	10.0
	AL	99.89	2.50	0.11	0.00	
	FL	97.77	2.50	0.81	1.43	
SVM	**NL**	71.84	2.50	0.01	28.14	14.5
	AL	84.52	2.50	0.13	15.35	
	FL	88.44	2.50	2.71	8.85	
RF	**NL**	99.94	2.50	0.03	0.03	12.74
	AL	99.73	2.50	0.26	0.01	
	FL	94.71	2.50	1.44	3.85	
Ensemble	**NL**	99.97	2.50	0.01	0.02	11.17
	AL	99.80	2.50	0.17	0.03	
	FL	94.68	2.50	4.36	0.96	

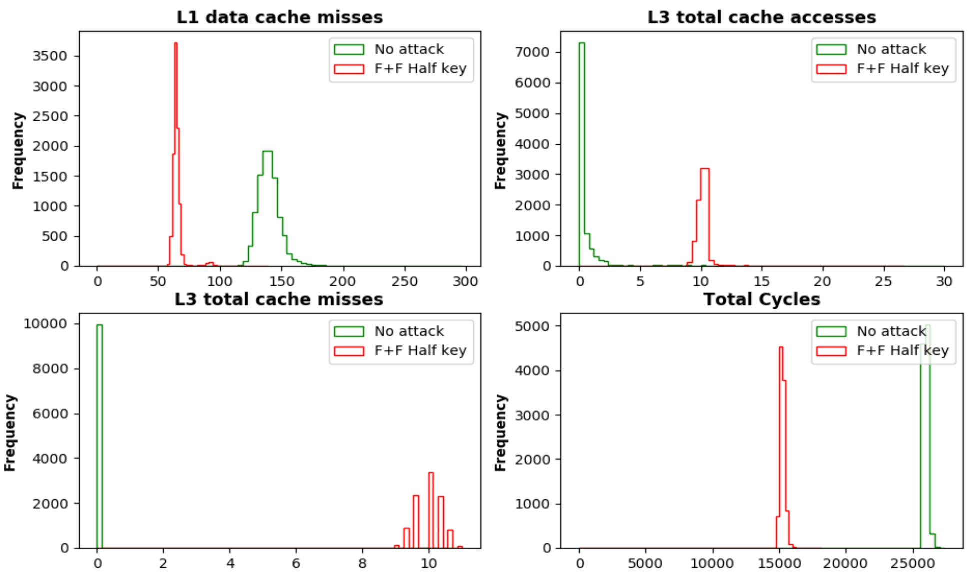

FIGURE 11: Run-time behavior of selected hardware events under No Load conditions for Flush+Flush *Impl.1*

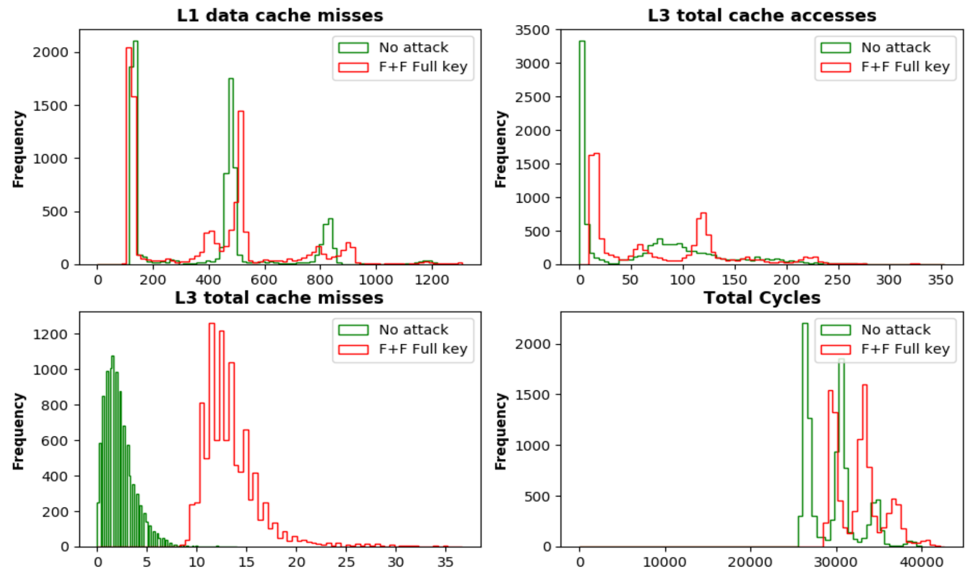

FIGURE 12: Run-time behavior of selected hardware events under Full Load conditions for Flush+Flush *Impl.2*

2) Detection Speed

A Flush+Flush attack on AES cryptosystem requires 350–400 encryptions for Impl.1 and $50,000$ encryptions for Impl.2 to extract the secret key. Our experimental results, shown in Tables 13 & 14, show that the WHISPER tool is capable of detecting Flush+Flush attack Impl.1 and Impl.2 within 2.5% and $0.02-0.04\%$ of their completion, respectively. That is, within the first $10-20$ encryptions out of $350-400$ and $50,000$ encryptions required by Flush+Flush attack Impl.1 & Impl.2, respectively. Like the first two case studies, this speed is achieved under variable load conditions under fine-grain detection granularity. Tables 15 & 16 show results for coarse-grain sampling, where the speed is reduced to 25% and 0.4% for Impl.1 and Impl.2, respectively, in the worst case. In terms of performance overhead, we observe a similar decreasing pattern as in other case studies in both implementations. With a Flush+Flush attack, the misclassification rate is generally higher than with the other two attacks. This is mainly due to the stealthy nature of this particular attack, which makes the models to misclassify more often. Here, we only analyze the reduction in the misclassification rate between fine-grain and coarse-grain detection scenarios

TABLE 14: Results using individual and Ensemble ML models for detection of Flush+Flush (*Impl.2*: full-key recovery) on AES at fine-grain sampling.

ML Model	Load	Accuracy (%)	Speed (%)	FP (%)	FN (%)	Overhead (%)
DT	NL	99.56	0.02	0.01	1.43	12.5
	AL	99.12	0.02	0.51	0.37	
	FL	97.84	0.02	1.54	0.62	
SVM	NL	72.86	0.02	0.02	27.12	11.5
	AL	87.44	0.04	0.50	12.06	
	FL	79.53	0.02	1.58	18.89	
RF	NL	98.16	0.02	0.08	1.76	14.78
	AL	98.80	0.02	0.95	0.25	
	FL	95.16	0.02	2.76	2.08	
Ensemble	NL	97.07	0.02	0.01	2.92	18.1
	AL	95.71	0.02	4.08	0.21	
	FL	94.82	0.02	1.76	3.42	

that are linked with detection speed. Overall, the misclassification rate decreases in the case of coarse-grain detection. Our results show that the magnitude of both FPs and FNs is reduced with coarse-grain sampling.

TABLE 15: Results using individual and Ensemble ML models for detection of Flush+Flush (*Impl.1*: half-key recovery) on AES at coarse-grain sampling.

ML Model	Load	Accuracy (%)	Speed (%)	FP (%)	FN (%)	Overhead (%)
DT	NL	99.97	25	0.02	0.00	0.2
	AL	99.79	25	0.20	0.00	
	FL	96.65	25	2.48	0.88	
SVM	NL	71.99	25	0.00	28.01	2.74
	AL	93.58	25	0.94	5.49	
	FL	91.43	25	1.50	7.07	
RF	NL	98.79	25	0.01	1.20	1.72
	AL	95.37	25	1.19	3.45	
	FL	96.07	25	2.88	1.05	
Ensemble	NL	98.79	25	0.01	1.20	2.96
	AL	95.54	25	1.01	3.45	
	FL	96.18	25	2.48	1.34	

3) Confusion Matrix

For a Flush+Flush attack, we analyze the misclassification error rate based on the same pattern as for Prime+Probe and Flush+Reload attacks. However, in this case, our findings are different. Tables 13 & 15 show our results concerning the error distribution as a percentage of FPs and FNs for Flush+Flush attack Impl.1 and Tables 14 & 16 show similar results for Impl.2. In case of Flush+Flush Impl.1, our experiments yield that individual models miss-classify with a significant high rate. For instance, SVM miss-classifies

TABLE 16: Results using individual and Ensemble ML models for detection of Flush+Flush (*Impl.2*: full-key recovery) on AES at coarse-grain sampling.

ML Model	Load	Accuracy (%)	Speed (%)	FP (%)	FN (%)	Overhead (%)
DT	NL	88.01	0.2	0.02	11.97	2.9
	AL	93.73	0.2	0.28	5.99	
	FL	90.31	0.2	1.80	7.89	
SVM	NL	74.07	0.4	0.00	25.93	4.13
	AL	90.68	0.2	2.19	7.12	
	FL	83.14	0.4	4.87	12	
RF	NL	87.32	0.4	0.03	12.65	5.46
	AL	92.71	0.2	2.48	4.81	
	FL	92.64	0.4	2.78	4.58	
Ensemble	NL	87.28	0.4	0.03	12.69	5.82
	AL	92.58	0.2	2.22	5.20	
	FL	91.72	0.4	2.47	5.81	

between 2.71%–1.50% as FPs and 28.14%–28.01% as FNs. The DT model miss-classifies between 0.81%–2.48% as FPs and 1.43%–0.88% as FNs. Similarly, the RF model miss-classifies between 1.44%–2.88% as FPs and 3.85%–3.45% as FNs in the worst-case. The Ensemble model, under the similar conditions, miss-classifies between 4.36%–2.48% as FPs and 0.96%–1.34% as FNs in the worst-case. Similar pattern can be found in case of Flush+Flush attack Impl.2 in Tables 14 & 16.

These results show that, for a given cryptosystem, depending on the target attack type and load conditions, some models may respond differently than others when used stand alone thus leading to much higher error rates in certain conditions. The use of an Ensemble model helps to maintain consistency in detection despite these variations and eliminates the impact of an individual model's error rate on overall accuracy under specific conditions. For instance, in this case, the effect of the SVM model error rate in detecting Flush+Flush attack has been eliminated thanks to the use of Ensemble by the WHISPER tool. The overall error in classification for the Ensemble model remains low and the miss-classification mainly constitutes False Positives. There is a small fraction of False Negatives, between 0.96%–1.34%, generated by Ensemble which is attributed to the stealthy nature of Flush+Flush attack. Yet, the overall distribution of error is well-managed by the Ensemble model.

4) Performance Overhead

Tables 13 & 14 show that the AES cryptosystem experiences a maximum slowdown of 14.5% and 14.7% while detecting Flush+Flush attack Impl.1 and Impl.2, respectively, under fine-grain detection granularity. This overhead is significantly reduced to maximum of 2.7% and 5.4% under coarse-grain detection for Impl.1 and Impl.2, respectively, as indicated in Tables 15 & 16. Overall, the performance overhead is comparatively large for a Flush+Flush attack compared to Prime+Probe and Flush+Reload attacks. This variation mainly originates from the implementation of the attack, which could have different make-span and thus lead to additional (or fewer) iterations of the victim's process, causing the detection module to execute more often.

VI. SCALABILITY OF WHISPER -THE CASE OF COMPUTATIONAL ATTACKS

Section V has provided a detailed experimental evaluation of the WHISPER tool against a large set of known side-channel attacks. In this section, we demonstrate that the proposed tool is also capable of detecting a diverse attack vector, which is not necessarily known a priori, i.e., for which the tool is not trained beforehand. We consider both a trivial scenario in which the tool uses its training on 02 CSCAs (Flush+Flush and Prime+Probe) to detect another CSCA (Flush+Reload), as well a non-trivial scenario by training WHISPER tool with a set of CSCAs (i.e., Flush+Reload, Flush+Flush and Prime+probe attacks) and try to detect other attacks (i.e., Spectre [21] and Meltdown [22] attacks), which are computational attacks and unknown to the tool. The later evaluation scenario inherently covers the trivial case. However, do provide results for both scenarios in Section VI-A.

Attacks like Spectre [21] and Meltdown [22] exploit out-of-order and speculative execution techniques that are available in almost all modern processors to maximize the utilization of execution units in CPU cores. Unlike Flush+Reload and Flush+Flush, these attacks are independent of the cryptosys-tem and can retrieve data from the victim's address space without having access privilege. This section briefly elaborates the scalability of the WHISPER tool by detecting both Spectre and Meltdown attacks. We demonstrate the scalability under two scenarios. In the first scenario, no modification is performed to WHISPER, i.e., no new features (hardware or software events) are added and no retraining of the ML models is performed. In the second scenario, we show that adding new features to WHISPER that are specifically related to the computational part and retraining the ML models can yield even better results. In the following, first we briefly explain the attack vector. In modern processors, when the control flow of the application depends on the result of a preceding instruction, the processor can predict the most likely path of the program and speculatively execute the next instructions. Depending on the size of the reorder buffer, speculative execution can run several hundreds of instructions ahead. In practice, it is known that speculative execution can lead to the incorrect execution of a program, but the CPU is designed to revert the results of incorrect speculative executions by simply not committing such results. Therefore, these errors were assumed to be safe prior to the discovery of Meltdown and Spectre attacks. However, it turns out that not all side effects of speculative execution are revertible and some previously leaked information, for instance, cache contents, can survive the revision of the CPU state. The Spectre and Meltdown attacks exploit this flawed behaviour by recovering the leaked information from the cache using CSCAs like Flush+Reload and Prime+Probe *etc*. Although Spectre and Meltdown attacks exploit computational optimization techniques like out-of-order and speculative execution, they still use CSCAs like Flush+Reload and Flush+Flush to potentially retrieve information from the caches.

The Spectre vulnerability exploits hardware features, such as branch prediction units, to reveal secret data. Speculative execution in CPUs, which results from branch mispredictions, leaves observable effects in the cache. Spectre vulnerability is verified as being effective on Intel, AMD and ARM processors. A Spectre attack is executed in two distinct phases: a branch instruction mistraining phase followed by a CSCA. Spectre attacks are reported to have many variants. However, the baseline vulnerability exploited by all variants is the mistraining of branch instructions, be it a branch direction predictor or Branch Target Buffer (BTB).

Similarly, Meltdown vulnerability exploits out-of-order and speculative execution. Out-of-order memory look-ups also leave noticeable effects in the cache, which are exploited by this particular attack. Meltdown vulnerability affects Intel processors and, according to ARM, some of their processors are also affected. Meltdown is also a two-phase attack that, in the first phase, bypasses memory isolation by unprivileged out-of-order execution and, in the second phase, performs CSCA to retrieve information from the caches that is being brought in but not committed due to exception. Meltdown generates segmentation faults or invalid page faults while it bypasses the memory isolation. Our experimental results show that Meltdown generates significantly more page faults than other processes.

Meltdown relies on a vulnerability specific to Intel and ARM processors and can be mitigated by implementating KAISER [21] in operating systems, whereas Spectre applies to vastly more CPU architectures and cannot be mitigated as effectively. To the best of our knowledge, no research work has reported on the detection of Meltdown attacks prior to this work. Some research work has, however, reported the detection of a Spectre attack and its variants [64]. Further details on how Spectre and Meltdown attacks work are provided in [21] and [22], respectively.

For our experiments, we use Perf and PAPI libraries to extract events related to Meltdown and Spectre attacks, respectively. Although Spectre attack have multiple variants we use one of the variants proposed in the original work on Spectre attack [21]. Since both attacks are independent of the cryptosystem, their detection therefore requires monitoring of all concurrent processes. We wrap the events around all active processes to sample the features at a coarse-grain frequency of 100 ms and demonstrate the detection results. Thus, the performance overhead in this case comprises the sampling of events for all processes and are therefore slightly higher (Tables 20 & 21).

A. SCENARIO 01: WITHOUT MODIFICATIONS IN THE WHISPER TOOL

It is pertinent to mention here that, in this case, the tool uses exactly the same HPCs and machine learning models without being retrained. That is, the tool has used its previous one-time training on cache-based SCAs to detect other attacks. At first, we have performed experiments with a trivial case in which, we have performed cross validation of the tool by training it with two CSCAs (Flush+Flush and Prime+Probe) and evaluated against an unknown CSCA (Flush+Reload), i.e., without training the tool for Flush+Reload attack. Table 17 shows that the detection accuracy is very high as anticipated intuitively. There were also no false negatives reported in this case.

TABLE 17: Results on the detection of Flush+Reload Attack using the WHISPER tool with training on only 02 attacks (Flush+Flush and Prime+Probe)

ML Model	Load	Accuracy (%)	Speed (ms)	FP (%)	FN (%)	Overhead (%)
DT	**NL**	98.08	100	1.92	0.0	0.51
	AL	96.435	100	3.565	0.00	
	FL	92.77	100	7.23	0.00	
SVM	**NL**	98.80	100	1.20	0.00	0.77
	AL	96.35	100	3.65	0.00	
	FL	94.12	100	5.88	0.00	
RF	**NL**	98.48	100	1.52	0.00	0.46
	AL	95.35	100	4.65	0.00	
	FL	89.57	100	10.43	0.00	
Ensemble	**NL**	96.88	100	3.12	0.00	0.79
	AL	93.435	100	6.565	0.00	
	FL	88.76	100	11.24	0.00	

Then, in a rather non-trivial case, the tool has used its previous one-time training on cache-based SCAs to detect Spectre and Meltdown attacks. This is relatively difficult execution scenario as these two attacks are no bache-based SCAs and the tool has no a priori knowledge of their behavior. Intuitively, retraining ML models involving variants of Spectre and Meltdown attacks would increase the detection accuracy and further reduce the chances of detection inconsistencies (discussed in Section VI-B).

Tables 18 and 19 present our detection results on Meltdown and Spectre attacks, respectively, using the WHISPER tool. Although the performance of WHISPER is evaluated using the same evaluation metrics, in these cases, sampling is adjusted to a fixed interval of 100 ms rather than a percentage of attack completion. This is because both these attacks are independent of cryptosystems. Therefore, they do not have a single victim process. Moreover, the overhead mainly constitutes the sampling cost of HPCs and not of any victim's slowdown. With a rather fixed and coarse-grain sampling rate, the overhead for the detection module is much less than in previous cases. As shown in Table 17, detection accuracy of all individual ML models under NL and AL conditions remains very high and the Ensemble consequently also provides very high accuracy. In the FL condition, however, the detection accuracy suffers and the tool generates a large number of False Positives. The fact that WHISPER is able to achieve reasonably good detection accuracies under 2 out of 3 load conditions is because both Spectre and Meltdown attacks use CSCAs to extract information from caches, and their behavior thus resembles that of the conventional CSCAs, which WHISPER is able to capture with no further training. An important result in the case of Meltdown detection is the fact that there are no False Negatives in the system. Similar pattern in results can be observed in Table 19 for Spectre attack detection. In this case, the overhead is slightly higher than in the detection of Meltdown. Figures 13–16 illustrate the variations in previously selected HPCs under NL and FL for Spectre and Meltdown attacks, respectively.

TABLE 18: Results on the detection of Meltdown Attack using the WHISPER tool without any modifications

ML Model	Load	Accuracy (%)	Speed (ms)	FP (%)	FN (%)	Overhead (%)
DT	**NL**	96.428	100	3.57	0.0	0.49
	AL	97.435	100	2.56	0.00	
	FL	42.857	100	57.14	0.00	
SVM	**NL**	92.857	100	7.14	0.00	0.70
	AL	97.435	100	2.56	0.00	
	FL	87.142	100	12.85	0.00	
RF	**NL**	96.428	100	3.57	0.00	0.47
	AL	97.435	100	2.56	0.00	
	FL	42.857	100	57.14	0.00	
Ensemble	**NL**	96.428	100	3.57	0.00	0.79
	AL	97.435	100	2.56	0.00	
	FL	42.857	100	57.14	0.00	

TABLE 19: Results on the detection of Spectre Attack using the WHISPER tool without any modifications

ML Model	Load	Accuracy (%)	Speed (ms)	FP (%)	FN (%)	Overhead (%)
DT	**NL**	97.058	100	2.94	0.00	1.14
	AL	91.176	100	8.82	0.00	
	FL	42.857	100	57.14	0.00	
SVM	**NL**	100.0	100	0.00	0.00	0.99
	AL	94.117	100	5.88	0.00	
	FL	87.142	100	12.85	0.00	
RF	**NL**	97.058	100	2.94	0.00	1.28
	AL	94.117	100	5.88	0.00	
	FL	42.857	100	57.14	0.00	
Ensemble	**NL**	97.058	100	2.94	0.00	1.30
	AL	94.117	100	5.88	0.00	
	FL	42.857	100	57.14	0.00	

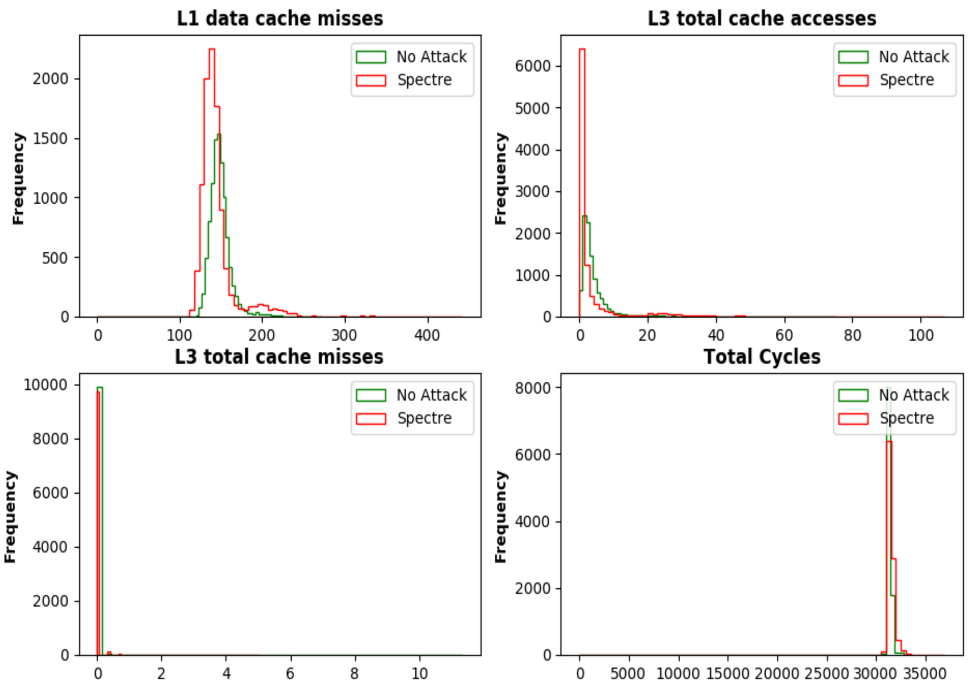

FIGURE 13: Run-time behavior of selected hardware events under No Load conditions for Spectre Attack

The significant variations in detection accuracy, particularly in the FL condition, originates from the fact that WHISPER did not use any additional training on the data set generated for Meltdown and Spectre attacks. Rather, it uses its previous training on Flush+Flush, Flush+Reload and Prime+Probe attacks to also detect Spectre and Meltdown, which become unknown attacks for WHISPER in this case.

B. SCENARIO 02: WITH MODIFICATION IN THE WHISPER TOOL

Section VI-A described results obtained with no modifications to the WHISPER tool. Intuitively, if the data set relevant to Spectre and Meltdown attacks is used in the training phase, the high rate of FPs in the previous case would naturally be reduced. Moreover, the use of new features that would make it possible to capture the behavior of these attacks more specifically can further enhance accuracy. Thus, we demonstrate that if WHISPER is enhanced with more specific features and retrained, its reported accuracy gets even better.

As mentioned earlier, the baseline vulnerability exploited by a Spectre attack is the mistraining of branch instructions, i.e., the branch direction predictor or BTB. Therefore, we select two new hardware events related to the Spectre attack: (1) Total Branch Instructions, and (2) Total Branch Mispredictions, as features for the ML models. Our experiments and analysis of Spectre attack code reveals that it has the much less number of total instructions to execute and, among those instructions, a large proportion of instructions comprise of branch instructions. Interestingly enough, the attack process generates the highest number of branch mispredictions. Our results show that, if total branch instructions and the total branch mispredictions generated by a process are collected as features, the ML models can detect Spectre attack with very high accuracy, as shown in Table 20 under all load conditions.

In the case of a Meltdown attack, similar observations can be made by observing the number of page faults generated by the attacker. Meltdown generates segmentation faults or invalid page faults while bypassing the memory isolation through exceptions. Page faults can be captured through a software event by using Perf API. Our experiments reveal that, for a relatively very small number of total instructions, the Meltdown attack process generates significantly large numbers of page faults that can serve as a very effective and highly uncorrelated feature for ML models in WHISPER. When we used page faults as one of the features, the detection accuracies against the Meltdown attack were significantly enhanced, as shown in Table 21.

VII. COMPARATIVE ANALYSIS AND DISCUSSION OF THE RESULTS–LESSONS LEARNED

Table 22 provides a brief comparative analysis of WHISPER with the state-of-the-art. We provide this comparison with respect to the evaluation metrics used in this work, *i.e.*, detection accuracy, speed, performance overhead and system load conditions in order to provide the reader with an overview of the existing techniques. Details on these techniques are already discussed in Section II. The authors in [59] detected Flush+Reload at an accuracy of F-score=0.93 and speed of

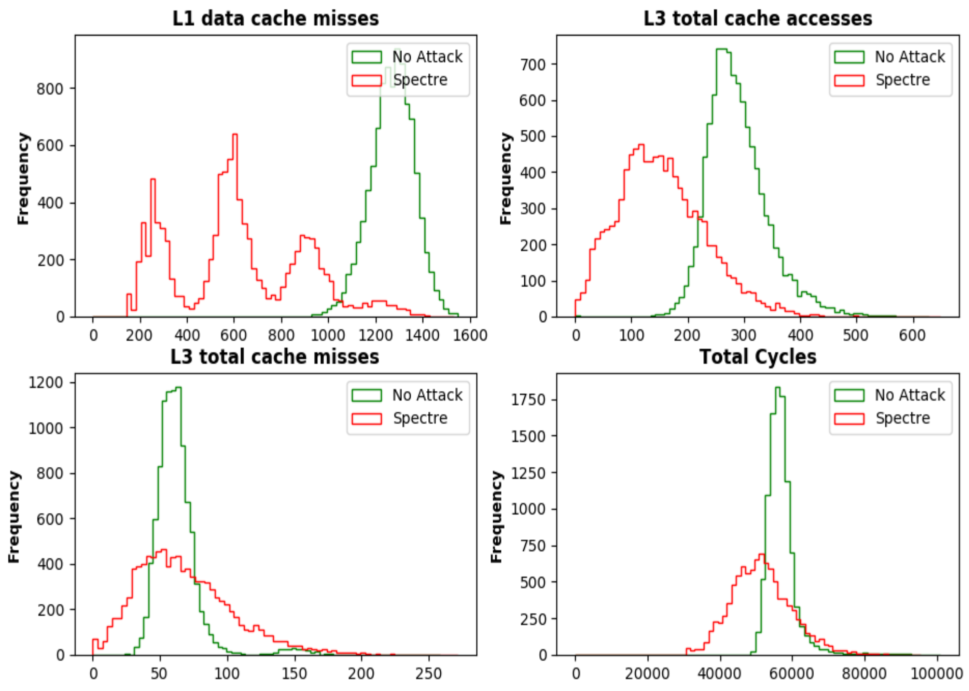

FIGURE 14: Run-time behavior of selected hardware events under Full Load conditions for Spectre Attack

TABLE 20: Results on the detection of Spectre attack using the WHISPER tool with modified features and retraining of ML Models.

Model	Loads	Accuracy (%)	Speed (ms)	FP (%)	FN (%)	Overhead (%)
DT	NL	99.93	100	0.07	0	1.6
	AL	99.06	100	0.57	0.37	
	FL	98.03	100	1.18	0.79	
RF	NL	99.97	100	0.03	0	1.6
	AL	98.40	100	1.27	0.33	
	FL	97.36	100	1.98	0.66	
SVM	NL	99.25	100	0.69	0.06	1.6
	AL	97.29	100	2.02	0.69	
	FL	95.87	100	2.87	1.26	
Ensemble	NL	99.80	100	0.17	0.03	1.6
	AL	99.13	100	0.57	0.29	
	FL	97.43	100	1.56	1.01	

TABLE 21: Results on the detection of Meltdown attack using the WHISPER tool with modified features and retraining of ML Models.

Model	Loads	Accuracy (%)	Speed (ms)	FP (%)	FN (%)	Overhead (%)
DT	NL	99.95	100	0.05	0	2.11
	AL	99.83	100	0.13	0.04	
	FL	98.27	100	1.24	0.49	
RF	NL	99.35	100	0.65	0	2.11
	AL	97.39	100	1.98	0.63	
	FL	94.67	100	3.43	1.90	
SVM	NL	99.97	100	0.03	0	2.11
	AL	99.17	100	0.67	0.16	
	FL	98.24	100	1.39	0.37	
Ensemble	NL	99.43	100	0.48	0.09	2.11
	AL	99.17	100	0.55	0.28	
	FL	98.69	100	0.79	0.52	

the 1/5th of attack completion in the presence of background noise. The F-score is a measure of accuracy in statistical analysis of binary classification. However, this technique does not discuss the impact on performance degradation.*HexPADS* [34] claims to have detected Flush+Reload and Prime+Probe attacks with 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. *CloudRadar* [24] claims to have detected Prime+Probe and Flush+Reload attacks with 100% accuracy and 5% overhead under No Load conditions as well. This technique also suffers from the same issues as *HexPADS*. The techniques proposed in [15] detect Flush+Reload and Prime+Probe attacks at 97% and 98% accuracy, respectively, and within 2% detection speed in the presence of background noise but did not discuss the over-

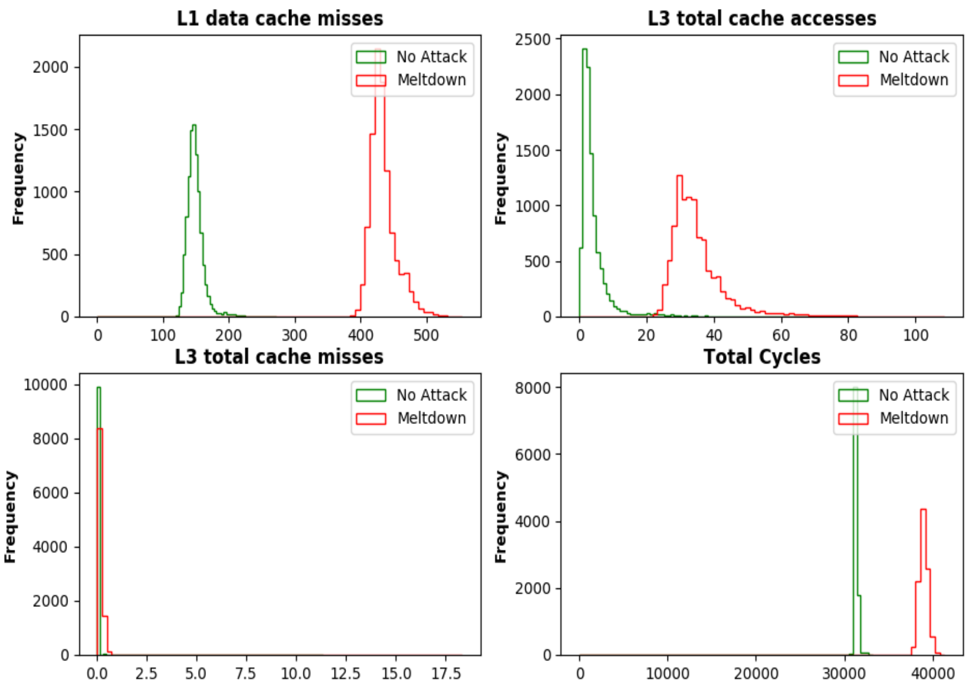

FIGURE 15: Run-time behavior of selected hardware events under No Load conditions for Meltdown Attack

head of their proposed mechanism. The technique proposed in [60] claims to detect template attacks with 99% accuracy but does not provide detection speed, overhead and any variations in system load conditions. The technique proposed in [35] claims to work 100% accurately on Prime+Probe attack with 2% overhead. However, the authors do not provide any result on the detection speed and the percentage of false negatives increases under load conditions with the proposed technique. The solution proposed in *SCADET* [33] claims to detect Flush+Reload at 100% accuracy in isolated conditions but there is no discussion about the impact of detection speed and overhead under load conditions. Similarly, The *SCADET* technique proposed in [33] claims to perform 100% accurately under different load conditions but the detection speed and the performance overhead of this mechanism are not discussed, on the other hand, it raises false alarms in load conditions and trace timing is long in some cases, which is not suitable for early stage detection.

The brief comparison provided in Table 22 supports our earlier discussion that detection accuracy alone is not a suit-able measure of a good detection technique. Based on these comparisons, our proposed detection tool clearly performs better under stringent evaluation metrics. Our mechanism works for a larger set of attacks while performing runtime detection. We provide results for multiple attacks, namely: Flush+Reload on AES, Flush+Flush on AES half-key (FF_Imp1), Flush+Flush on AES full-key (FF_Imp2), Prime+Probe on AES half-key (FF_Imp1), Prime+Probe on AES full-key (FF_Imp2), Spectre and Meltdown. The reported accuracy of our tool for these attacks is comparable or better than the state-of-the-art techniques. Similarly, the reported detection speed is well within the theoretical upper bound of 50% attack completion and performance overhead remains low under variable load conditions.

A. DISCUSSION

Throughout this work, we have built our argument in favor of detection-based protection, which would help apply mitigation only after successful detection of CSCAs. Our experiments applied to three different case studies, comprising the state-of-the-art CSCAs, demonstrate that detection can be highly accurate with a minimum system overhead at runtime and fast enough to raise the alarm before attack completion. Our experiments with different attack categories enabled us to provide an evidence-based analysis of CSCA detection.

The first lesson we learned from these results is that almost all CSCAs, whether they are known or unknown, dependent or in-dependent of cryptosystem, leave their footprints on the cache hierarchy, either in the form of access timing or access pattern. Such a footprint can be captured by carefully profiling the behavior of the affected process(es) without a priori knowledge of the type of attack or the temporal order of multiple attacks taking place. To this end, selection of most relevant hardware events is of paramount importance, as demonstrated in Section IV-B1. Nevertheless, it is pertinent to mention here that the underlying hardware events may be imprecise, non-deterministic and limited in number, which can lead to an increased error rate (FPs & FNs) under smarter attacks in the future. Das *et al.* [65] provide a detailed insight into the limitations and pitfalls of using HPCs for security.

We have provided the proof of concept on the use of WHISPER for a larger attack vector. For instance, we have shown detection results for cache-based attacks like Flush+Reload, Flush+Flush and

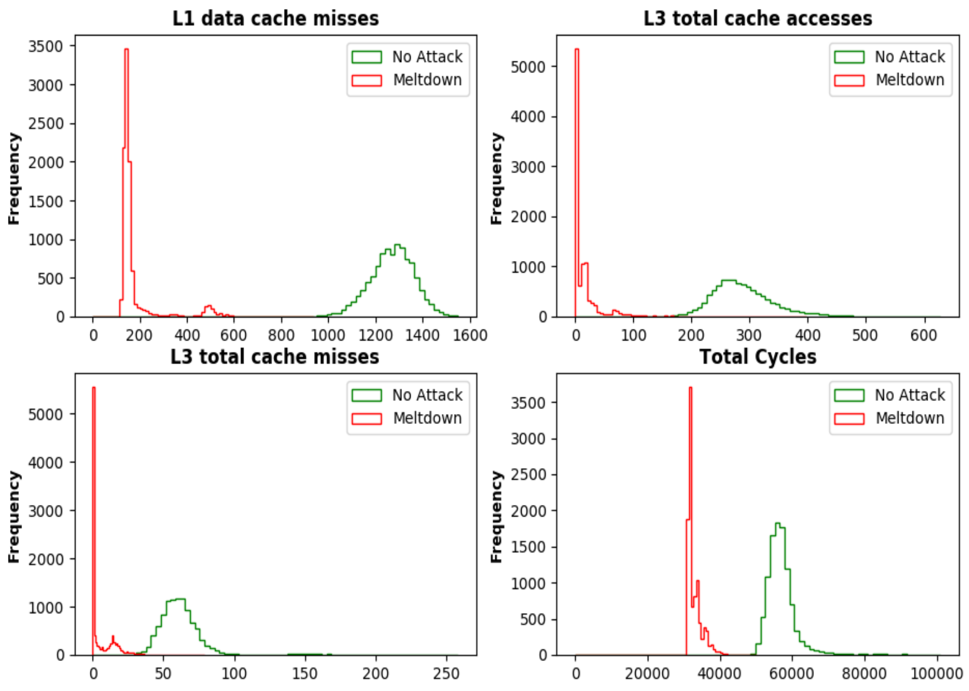

FIGURE 16: Run-time behavior of selected hardware events under Full Load conditions for Meltdown Attack

Prime+Probe as well as for computational attacks like Spectre Meltdown. We show that WHISPER is able to detect available state-of-the-art attacks that target cryptosystems as well as those that are independent of cryptosystems. Our results show significant high detection accuracy for these attacks with reasonably low performance overhead, making WHISPER a very good candidate for adoption for early-stage runtime detection.

The second lesson we learned from our experiments is that simple statistical or threshold-based solutions are not sufficient to distinguish anomalous behavior from normal behavior, particularly in the case of side-channel attacks. As shown in Section IV-C, the attacks can take place in any temporal order and the data being collected by the hardware events might not be easy to classify. In certain cases, even stand-alone ML models might not be sufficient to detect anomalous behavior, as illustrated in our third case study with The SVM model (Section V-D3). Our experiments with 12 different ML models and the use of Ensemble model provide empirical evidence to strengthen the belief that machine learning can help building the resilient software/hardware security solutions for modern computing systems. We have demonstrated their success on known CSCAs.

The third lesson we learned is that detection tools and techniques must be evaluated in a holistic manner, *i.e.*, by considering security as well as performance aspects. For instance, very high detection accuracy is not enough if the tool cannot be adopted for runtime detection due to its slow speed or considerably high runtime overhead, which would cause significant slowdown for the victim's process. The use of multiple stringent evaluation metrics in this work reveals that there is a trade-off between the performance overhead and the detection speed of a runtime detection tool. In order to serve as the first line of defense against SCAs, a detection tool must be fast enough to report an attack before its completion and yet it has to be light-weight enough to continuously monitor the system's behavior without significantly increasing the overhead. Our experiments show that increased detection granularity yields more reliable results in terms of speed, accuracy and misclassification rates, resulting in increased performance overhead. With decreased granularity, performance overhead is significantly reduced. Therefore, we propose to use the WHISPER tool in two different modes, *i.e.*, fine-grain and coarse-grain sampling modes. The tool offers this flexibility to run in either of these modes depending on the operating conditions and persisting threat levels. Once a threat is detected, the tool can adjust its sampling rate to confirm the detection and subsequently report to the OS to take remedial measures.

The fourth and last lesson we learned is that, unless there are design changes at the hardware level and a complete overhauling of the entire computing stack, software alone cannot completely protect against side-channel information leakage. It can only make such leakage harder. Detection solutions that are entirely based on software are vulnerable to adversarial attacks, which could corrupt software features for ML models in order to overcome the detection mechanism. From an in-depth analysis and experimentation in this work, we have learned that it is hard to corrupt the values of hardware events directly collected using HPCs at runtime. In order to do so, the malicious processes need to alter the behavior of caches at the hardware level to camouflage their activities, which is much harder than tempering software-based values. Therefore, hypothetically, even if the adversary knows the detection mechanism as a white box, it is still very hard to manipulate the information of hardware events at runtime.

VIII. CONCLUSIONS AND FUTURE WORK

This paper argued in favor of using runtime detection as the first line of defense against cache side-channel attacks. We advocate for a detection-based protection mechanisms as existing mitigation techniques against SCAs either completely remove or greatly reduce the performance benefits of resource sharing. In this paper, we propose a machine learning based CSCA detection tool, called WHISPER, for Intel $x86$ architecture. The tool comprises multiple machine learning models, integrated in an Ensemble fashion, that use real-

TABLE 22: Comparative analysis of WHISPER with the state-of-the-art [Note: F+R=Flush+Reload, F+F=Flush+Flush, P+P=Prime+Probe, NA=Not Available].

Ref.	Attack	Accuracy	Speed	Overhead	Load
[59]	F+R	F-score 0.93	1/5th of attack completion	NA	Yes (Apache)
[34]	F+R, P+P	100%	NA	<2%	No
[24]	P+P, F+R	100%	NA	<5%	No
[15]	F+R, P+P	97%, 98%	2%	NA	Yes (SPEC)
[32]	Template Attacks	>99%	NA	NA	No
[60]	P+P	100%	NA	2%	Yes (CIW)
[35]	F+R	100%	NA	NA	No
[33]	P+P	100%	NA	NA	Yes (Open source)
This work	F+R(AES), F+F(AES, Imp1), F+F(AES, Imp2), P+P(AES, Impl1), P+P(AES, Imp2), Spectre, Meltdown	99.99%, 99.99%, 99.8% , 99.99%, 99.77%, 97.05%, 97.43%	<40%, <25%, <25%, <0.21%, <0.21%, fixed, fixed	< 8%	Yes (SPEC)

time behavioral data of concurrent processes running on Intel $x86$ architecture. The WHISPER tool is capable of detecting a large set of the state-of-the-art attacks without the need to retrain its machine learning models for each specific type of attack. We describe extensive experimentation with six different attacks and evaluate the tool under stringent constraints, such as: detection accuracy, speed, performance overhead and distribution of error (*i.e.*, false positives and false negatives). Our results show very high detection accuracy, *i.e.*, $> 99\%$, with a negligible error rate. We have also demonstrated the scalability of WHISPER by detecting computational attacks, *i.e.*, Spectre and Meltdown. Results show that WHISPER, without retraining and no modification of the hardware events, is able to detect both vulnerabilities with a reasonable accuracy. The tool is light weight and easy to integrate for runtime detection. We provide experimental evaluation of the tool under variable load conditions to demonstrate its resilience and consistency in noisy environment. As a future direction, specialized machine learning models can be trained to detect partially known or fully unknown attacks.

REFERENCES

[1] W. Stallings, L. Brown, M. D. Bauer, and A. K. Bhattacharjee, "Computer security: principles and practice". Pearson Education Upper Saddle River, NJ, USA, 2012.

[2] Y. Yarom, D. Genkin, and N. Heninger, “CacheBleed: a timing attack on OpenSSL constant-time RSA,” Journal of Cryptographic Engineering, vol. 7, no. 2, pp. 99–112, 2017.

[3] Y. Yarom and K. Falkner, “FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack,” in 23rd USENIX Conference on Security Symposium, 2014.

[4] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, “Flush+Flush: A Fast and Stealthy Cache Attack,” in Proceedings of the 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment - Volume 9721, DIMVA 2016, 2016.

[5] D. A. Osvik, A. Shamir, and E. Tromer, "Cache Attacks and Countermeasures: The Case of AES". Springer Berlin Heidelberg, 2006.

[6] G. Irazoqui, M. S. Inci, T. Eisenbarth, and B. Sunar, “Wait a minute! a fast, cross-VM attack on AES,” in International Symposium on Research in Attacks, Intrusions, and Defenses, pp. 299–319, 2014.

[7] Q. Ge, Y. Yarom, D. Cock, and G. Heiser, “A survey of microarchitectural timing attacks and countermeasures on contemporary hardware,” Journal of Cryptographic Engineering, pp. 1–27, 2016.

[8] 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 Int’l Symposium on Parallel & Distributed Processing with Applications, pp. 519–524, Aug 2009.

[9] F. Liu and R. B. Lee, “Random Fill Cache Architecture,” in Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 203–215, 2014.

[10] T. Kim, M. Peinado, and G. Mainar-Ruiz, “STEALTHMEM: System-Level Protection Against Cache-Based Side Channel Attacks in the Cloud,” in USENIX Security, pp. 189–204, 2012.

[11] M. Chiappetta, E. Savas, and C. Yilmaz, “Real time detection of cache-based side-channel attacks using hardware performance counters,” Applied Soft Computing, vol. 49, pp. 1162–1174, 2016.

[12] 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 Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy, HASP ’18, (New York, NY, USA), pp. 1:1–1:8, ACM, 2018.

[13] M. Mushtaq, A. Akram, M. Bhatti, N. B. R. Rao, V. Lapotre, and G. Gogniat, “Run-time detection of Prime+Probe side-channel attack on AES encryption algorithm,” in Global Information Infrastructure and Networking Symposium, Greece, 2018.

[14] M. Mushtaq, A. Akram, M. K. Bhatti, M. Chaudhry, M. Yousaf, U. Farooq, V. Lapotre, and G. Gogniat, “Machine Learning For Security: The Case of Side-Channel Attack Detection at Run-time,” in 25th IEEE International Conference on Electronics Circuits and Systems, Bordeaux, FRANCE, 2018.

[15] Z. Allaf, M. Adda, and A. Gegov, “A comparison study on Flush+Reload and Prime+Probe attacks on AES using machine learning approaches,” UK Workshop on Computational Intelligence, 2017.

[16] Y. Lyu and P. Mishra, “A survey of side-channel attacks on caches and countermeasures,” Journal of Hardware and Systems Security, vol. 2, no. 1, pp. 33–50, 2018.

[17] M. S. Inci, B. Gulmezoglu, G. Irazoqui, T. Eisenbarth, and B. Sunar, “Cache attacks enable bulk key recovery on the cloud,” in International Conference on Cryptographic Hardware and Embedded Systems (CHES), vol. 9813, pp. 368–388, 08 2016.

[18] D. Gruss, R. Spreitzer, and S. Mangard, “Cache template attacks: Automating attacks on inclusive last-level caches,” in Proc. of 24th USENIX Conf. on Security Symp., (Berkeley, CA, USA), pp. 897–912, USENIX Assoc., 2015.

[19] J. V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, “FORESHADOW: Extracting the keys to the intel SGX kingdom with transient out-of-order execution,” in 27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018., pp. 991–1008, 2018.

[20] D. Genkin, L. Valenta, and Y. Yarom, “May the Fourth Be With You: A microarchitectural side channel attack on several real-world applications of Curve25519,” in Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017, pp. 845–858, 2017.

[21] P. Kocher, J. Horn, A. Fogh, , D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, “Spectre

attacks: Exploiting speculative execution," in 40th IEEE Symposium on Security and Privacy (S&P'19), 2019.
[22] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in 27th USENIX Security Symposium (USENIX Security 18), 2018.
[23] M. Godfrey and M. Zulkernine, "Preventing cache-based side-channel attacks in a cloud environment," IEEE Transactions on Cloud Computing, vol. 2, pp. 395–408, Oct 2014.
[24] 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, Springer, 2016.
[25] F. Liu, H. Wu, K. Mai, and R. B. Lee, "Newcache: secure cache architecture thwarting cache side channel attacks," IEEE Micro Special Issues on Security, vol. 36, 2016.
[26] Z. Wang and R. B. Lee, "New cache designs for thwarting software cache-based side channel attacks," SIGARCH Comput. Archit. News, vol. 35, pp. 494–505, June 2007.
[27] W.-M. Hu, "Reducing timing channels with fuzzy time," Journal of computer security, vol. 1, no. 3-4, pp. 233–254, 1992.
[28] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Cross-VM Side Channels and Their Use to Extract Private Keys," in ACM CCS, (NY, USA), 2012.
[29] V. Varadarajan, T. Ristenpart, and M. Swift, "Scheduler-based defenses against cross-VM side-channels," in 23rd USENIX Security Symposium, (San Diego, CA), 2014.
[30] C. Canella, J. Van Bulck, M. Schwarz, M. Lipp, B. Von Berg, P. Ortner, F. Piessens, D. Evtyushkin, and D. Gruss, "A systematic evaluation of transient execution attacks and defenses," in 28th USENIX Security Symposium (USENIX Security 19), pp. 249–266, 2019.
[31] G. Irazoqui, K. Cong, X. Guo, H. Khattri, A. K. Kanuparthi, T. Eisenbarth, and B. Sunar, "Did we learn from LLC side channel attacks? A cache leakage detection tool for crypto libraries," CoRR, vol. abs/1709.01552, 2017.
[32] M. Alam, S. Bhattacharya, D. Mukhopadhyay, and S. Bhattacharya, "Performance counters to rescue: A machine learning based safeguard against micro-architectural side-channel-attacks." Cryptology ePrint Archive, Report 2017/564, 2017. https://eprint.iacr.org/2017/564.
[33] M. Sabbagh, Y. Fei, T. Wahl, and A. A. Ding, "SCADET: a side-channel attack detection tool for tracking Prime+Probe," in 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2018.
[34] M. Payer, HexPADS: A Platform to Detect "Stealth" Attacks, pp. 138–154. Cham: Springer International Publishing, 2016.
[35] 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.
[36] A. Raj and J. Dharanipragada, "Keep the PokerFace on! Thwarting cache side channel attacks by memory bus monitoring and cache obfuscation," Journal of Cloud Computing, vol. 6, no. 1, p. 28, 2017.
[37] C. Maurice, C. Neumann, O. Heen, and A. Francillon, "C5: cross-cores cache covert channel," in International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, pp. 46–64, Springer, 2015.
[38] A. Barresi, K. Razavi, M. Payer, and T. R. Gross, "CAIN: Silently Breaking ASLR in the Cloud," in 9th USENIX Workshop on Offensive Technologies (WOOT), 2015.
[39] M. Seaborn and T. Dullien, "Exploiting the dram rowhammer bug to gain kernel privileges," Black Hat, vol. 15, p. 71, 2015.
[40] S. Chen, X. Zhang, M. K. Reiter, and Y. Zhang, "Detecting privileged side-channel attacks in shielded execution with déjá vu," in Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, pp. 7–18, ACM, 2017.
[41] S. Briongos, G. Irazoqui, P. Malagón, and T. Eisenbarth, "CacheShield: Detecting cache attacks through self-observation," in Proceedings of the 8th Conference on Data & Application Security & Privacy, pp. 224–235, ACM, 2018.
[42] Y. Kulah, B. Dincer, C. Yilmaz, and E. Savas, "SpyDetector: An approach for detecting side-channel attacks at runtime," IJIS, 2018.
[43] "Online repository of cache side-channel attacks," https://github.com/ECLab-ITU, 2019.
[44] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in Proceedings of the 2015 IEEE Symposium on Security and Privacy, SP '15, (Washington, DC, USA), pp. 605–622, IEEE Computer Society, 2015.
[45] Nepoche, "https://github.com/nepoche/flush-reload," 2017.
[46] D. Gruss, "https://github.com/iaik/flush_flush," 2017.
[47] E. Tromer, D. A. Osvik, and A. Shamir, "Efficient Cache Attacks on AES, and Countermeasures," Journal of Cryptology, vol. 23, no. 1, pp. 37–71, 2010.
[48] C. Percival, "Cache missing for fun and profit," in Proc. of BSDCan 2005, 2005.
[49] O. Aciicmez, B. B. Brumley, and P. Grabher, "New results on instruction cache attacks," in Proc. of Int'l Conference on Cryptographic Hardware and Embedded Systems, CHES'10, (Heidelberg), pp. 110–124, Springer-Verlag, 2010.
[50] O. Aciicmez and W. Schindler, "A vulnerability in RSA implementations due to instruction cache analysis and its demonstration on OpenSSL," in Proceedings of the 2008 The Cryptopgraphers' Track at the RSA Conference on Topics in Cryptology, CT-RSA'08, (Berlin, Heidelberg), pp. 256–273, Springer-Verlag, 2008.
[51] B. C. Vattikonda, S. Das, and H. Shacham, "Eliminating fine grained timers in Xen," in CCSW, 2011.
[52] "Intel 64 and ia-32 arch. software developer's manual volume 3b: System programming guide, part2.," June 2014.
[53] C. M. Bishop, Pattern Recognition and Machine Learning (Information Science and Statistics). Berlin, Heidelberg: Springer-Verlag, 2006.
[54] PerfMon, "$https : //knowledge.ni.com/$," 2018.
[55] OProfile, "$http : //oprofile.sourceforge.net/$," 2018.
[56] P. Tool, "$http : //lacasa.uah.edu/$," 2018.
[57] I. V-Tune, "$https : //software.intel.com/en - us/vtune - amplifier - cookbook$," 2018.
[58] "Performance application programming interface," in $http : //icl.cs.utk.edu/papi/$, 2018.
[59] M. Chiappetta, E. Savas, and C. Yilmaz, "Real time detection of cache-based side-channel attacks using hardware performance counters," Journal of Applied Soft Computing, vol. 49, pp. 1162–1174, Dec. 2016.
[60] 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 Fog and Mobile Edge Computing (FMEC), 2018 Third International Conference on, pp. 7–12, IEEE, 2018.
[61] O. Chapelle, B. Scholkopf, and A. Zien, "Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]," IEEE Transactions on Neural Networks, vol. 20, no. 3, pp. 542–542, 2009.
[62] G. James, D. Witten, T. Hastie, and R. Tibshirani, An introduction to statistical learning, vol. 112. Springer, 2013.
[63] J. Friedman, T. Hastie, and R. Tibshirani, The elements of statistical learning, vol. 1. Springer series in statistics New York, 2001.
[64] J. Depoix and P. Altmeyer, "Detecting spectre attacks by identifying cache side-channelattacks using machine learning," in WAMOS 2018 Fourth Wiesbaden Workshop on Advanced Microkernel Operating Systems, Wiesbaden, Germany., Hochschule RainMan, Computer Engineering Department, August 2018.
[65] S. Das, J. Werner, M. Antonakakis, M. Polychronakis, and F. Monrose, "Sok: The challenges, pitfalls, & perils of using hardware performance counters for security," IEEE Symposium on Security & Privacy, 2019.

MARIA MUSHTAQ Maria Mushtaq is a CNRS Post Doctoral researcher at LIRMM, University of Montpellier (UM), France. She did her Ph.D. from Lab-STICC, University of South Brittany (UBS), France in 2019. Maria has specific expertise in developing runtime detection and mitigation solutions against side-channel information leakage in computing systems. Her research interests mainly focus on cryptanalysis, constructing and validating software security components, and constructing OS-based security primitives against various hardware vulnerabilities.

JEREMY BRICQ Jeremy Bricq is a PhD student from the Université Catholique de Louvain, Belgium, since 2019. He spent 3 months internship in the Université de Bretagne-Sud, France, where he worked on the Side-Channel Attacks detection. Jeremy received a Master in Cybersecurity from the Université Libre de Bruxelles. His current works focus on Cryptography, with Fully Homomorphic Encryption.

MUHAMMAD KHURRAM BHATTI is an Assistant Professor at Information Technology University, Lahore, Pakistan. He did his Postdoc from KTH Royal Institute of Technology, Stockholm, Sweden. He did his MS and Ph.D. in Embedded Systems from University of Nice-Sophia Antipolis, France. His research interests include Embedded Systems, Information Security at both hardware and software level, Cryptanalysis, Mixed Criticality and Parallel Computing Systems.

AYAZ AKRAM Ayaz Akram received his bachelor's degree in electrical engineering from the University of Engineering and Technology, Lahore, Pakistan, and his master's degree in computer engineering from Western Michigan University, USA. He is currently pursuing a Ph.D. degree in computer science with the University of California at Davis, USA. His research interests include computer architecture, high-performance computing, and the intersection of computer architecture with other areas like computer security and machine learning.

VIANNEY LAPOTRE Vianney LAPOTRE received his MSc and PhD in Electrical and Computer Engineering from the University Bretagne-Sud, France. He spent six months as an invited researcher at the Ruhr-University of Bochum, Germany. He was a Postdoctoral at LIRMM, Montpellier, France. He is an associate professor at University Bretagne-Sud, France. His research interests include hardware security, reconfigurable and self-adaptive multiprocessor architectures.

GUY GOGNIAT is a Professor in ECE with the University of Bretagne-Sud, Lorient, France, where he has been since 1998. In 2005, he spent one year as an invited Researcher with the University of Massachusetts, Amherst, USA, where he worked on embedded security using reconfigurable technologies. His work focuses on embedded systems design methodologies and tools. He also conducts research in the domain of reconfigurable and adaptive computing and embedded system security.

PASCAL BENOIT received the Ph.D. degree in microelectronics from the University of Montpellier, France, in 2004, and the Habilitation degree from the Montpellier Laboratory of Informatics, Robotics and Microelectronics, University of Montpellier, in 2015. He was a Scientific Assistant with the Karlsruhe Institute of Technology, University of Karlsruhe, Germany. Since 2005, he has been a Permanent Associate Professor with the Montpellier Laboratory of Informatics, Robotics and Microelectronics, University of Montpellier. He has co-authored over 130 publications in books, journals, and conference proceedings. He holds five patents. His research interests include the Internet of Things, from smart sensors to gateways, energy efficiency, and security issues.