Hybrid-Shield: Accurate and Efficient Cross-Layer Countermeasure for Run-Time Detection and Mitigation of Cache-Based Side-Channel Attacks

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

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

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

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

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

ABSTRACT

Cache-based Side-Channel Attacks (SCAs) exploit the emerging hardware vulnerabilities to steal secret information by observing cache access patterns of cryptographic applications. To address the challenges introduced by SCAs, existing solutions either rely on detecting them using the profiled hardware-related information of victim and attack programs, or mitigating the cache-based SCAs by focusing on cache partitioning techniques and/or randomization of cache mappings by modifying the underlying cache architecture of the processor. However, the former's detectors highly rely on the knowledge of attack programs that are not always available or could be obfuscated in real-world benign applications resulting in misidentifying attacks. On the other hand, the latter approach though effective, requires modification in the processor hardware which increases the complexity of architecture design and are not applicable to current as well as legacy architectures. To address the drawbacks, we propose *Hybrid-Shield*, an accurate and efficient cross-layer countermeasure for run-time detection and mitigation of cache-based side-channel attacks. For the detection stage, microarchitectural information of victim under attack and under no attack conditions are collected for training machine learning classifiers. For the mitigation stage, *Hybrid-Shield* adapts hardware prefetchers and scales processor frequency to increase the noise level in observed cache access pattern attacks to induce secret information. The experimental results indicate that *Hybrid-Shield* can achieve 100% detection rate with 0% false alarm rate and detected attacks' error rate increases from less than 5% to above 35% with only 15% performance overhead.

KEYWORDS

Side-Channel Attacks, Machine Learning, Detection, Mitigation

ICCAD '20, November 2–5, 2020, Virtual Event, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-8026-3/20/11…$15.00
https://doi.org/10.1145/3400302.3418783

1 INTRODUCTION

Side-Channel Attack (SCA) is a class of attacks that observe "side-channel" information and result in the leakage of confidential data. The cache-based SCAs are exploiting the cache hierarchy and observing cache access patterns of cryptographic applications [31, 35]. Thus, they are emerging as a severe threat to the security of computer systems. Recent developed cache-based SCAs [7, 16, 20, 28, 39, 40] can be launched by the attacker remotely. Such attacks exploit the accessing time gap between the on-chip caches and main memory, and collect cache hit/miss traces based on different accessing times. Based on cache patterns observed, the attacks deduce sensitive information. Due to the invisibility, feasibility, and capability to expose and extract the secret keys in the cache-based SCAs [16, 40], there is an urgent need to address the security risks posed by such attacks in the present computer systems as well as legacy systems. To address the challenges brought by SCAs, researchers have proposed different techniques to detect or mitigate the attacks to prevent the computer systems from being compromised.

Prior works on side-channel attacks detection such as [5, 8, 10, 18, 25, 30, 34, 36, 42] proposed using microarchitectural pattern analysis captured through Hardware Performance Counters (HPCs) to detect the SCAs with latency ranging from milliseconds to seconds. Furthermore, cache-based SCAs mitigation can be categorized into two main design principles, including cache partitioning [12, 19, 29] and randomization-based techniques [21, 35, 37, 38]. Cache partitioning methods have been proposed to isolate cache usage between different programs to prevent attackers from influencing and observing the victims' data access patterns [12, 19, 29, 38]. Whereas, randomization techniques [21, 37] attempt to randomize the memory-to-cache mapping by adding random noises to cache observations to mitigate the impact of SCAs.

In this paper, we have identified and addressed major challenges of realizing an efficient run-time SCAs detection and mitigation approach which have not been considered in prior works:

❶ **Lack of Robustness:** Our comprehensive analysis shows that the previous works on SCAs detection jointly correlate the HPCs traces of victim and attack applications [8, 42] where the detectors require HPCs data from both the attack and victim application. However, recent studies [2, 11, 15, 17] show that attackers can craft

Table 1: Comparison of the recent works on SCAs protection (Red color indicates drawbacks of the work)

Prior Works	Detection Metrics				Mitigation Metrics		
	HPCs Evaluation	Required SCAs' Code	Latency	False Alarm Solution	New Architecture	Performance Overhead	Application Modification
CloudRadar [42]	Yes	Yes	100s-5000s	No	No mitigation module		
Real-Time Detection [9]	No	Yes	1.5 2.4s	No	No mitigation module		
Nights-watch[25]	No	No	No Mentioned	No	No mitigation module		
Cacheshield[3]	No	No	11 ms-24ms	No	No mitigation module		
CPU Elasticity [23]	No detection module				No	32.66%	No
FLUSH+PREFETCH[24]	No detection module				No	Not Mentioned	No
Random Fill[21]	No detection module				Yes	No Mentioned	No
Catalyst[19]	No detection module				Limited Architectures	0.16%-17.6%	No
★ Hybrid-Shield	Yes	No	500 μs−5ms	Yes	No	15%	No

attacks to bypass detectors. In addition, hiding SCAs in benign applications also helps to collect HPCs dataset for detection. Hence, current detection techniques relying on the HPCs of attack applications are facing significant security threats since the HPCs of SCAs can be easily manipulated or hidden.

❷ **Comprehensive Study of Machine Learning Classifiers:** A wide range of classification techniques can be developed by applying off-the-shelf Machine Learning (ML) algorithms. However, existing works in particular on SCAs detection have primarily focused on one or limited ML techniques for attacks detection and classification [8, 9, 25, 42]. Comprehensive analysis of diverse classifiers for SCAs detection is important as each could yield different performance (in terms of accuracy, false positive rate, computational complexity, etc.) [32, 34].

❸ **High False Alarm Rate:** Prior studies on real-time HPC-based SCAs detection have neglected to examine the instance level (a complete temporal sequence of victims' HPCs) false positives and have only evaluated the SCAs detectors based on the interval level (a sub-sequence of victims' HPCs) false positive rate [3, 8, 42]. However, SCAs detection based on the capturing intervals of HPCs data is biased to ”under attack” condition. The Victim under No Attack (VNA) requires all captured HPCs intervals to be classified correctly by the SCAs detector to achieve a correct prediction, while the Victim under Attack (VA) requires only one interval classified correctly to achieve a correct prediction.

❹ **High Overhead and Hardware Modification:** Our analysis indicates that existing works on mitigating cache-based side-channel attacks have mainly focused on cache partitioning and/or randomization techniques. However, such solutions require redesign effort of computer architectures [21, 22]. While the work in [23] does not incur new design, it results in considerable performance overhead to the system.

In response to the challenges mentioned above, this work proposes the *Hybrid-Shield*, an accurate and efficient cross-layer countermeasure for run-time detection and mitigation of cache-based side-channel attacks. For the detection stage, microarchitectural information of victim under attack and under no attack conditions is collected for training and testing machine learning (ML) classifiers. The ML-based detectors are then deployed to facilitate an effective run-time SCAs detection using only the HPCs features of victim application with high prediction accuracy and low instance level false alarm rate. For the mitigation stage, *Hybrid-Shield* offers a lightweight system and architecture level randomization technique to effectively mitigate the security threat from SCAs with no hardware redesign overhead. To this aim, by carefully adapting the processor frequency and prefetchers, a proper level of noise is added to the attackers' cache observations, protecting the critical information from being leaked. Table 1 characterizes the existing countermeasure techniques to combat the challenges of SCAs and further highlights the contributions of our proposed work. *Hybrid-Shield* eliminates the need for side-channel attacks' HPCs information, supports an effective false alarm minimization method, and does not require additional architecture modification for SCAs mitigation with less performance overhead.

Contributions. The main contributions of this work are summarized as follows:

- To eliminate the influence of untrustworthy attack's HPCs, *Hybrid-Shield* first attempts to detect SCAs based on only victim applications under two conditions: 1) Victim under Attack (VA), and 2) Victim under No Attack (VNA).
- Various ML classification algorithms are explored to find the most accurate classifier for detecting SCAs.
- A customized set of HPC features and False Alarm Minimization (FAM) are proposed to improve the detection accuracy while lowering the false alarm rate.
- *Hybrid-Shield* is further equipped with a comprehensive system and architecture level randomization solution to mitigate the SCAs on last-level caches efficiently.

2 MOTIVATIONAL CASE STUDIES

2.1 ML based SCAs Detector with Victim HPCs

- *Unreliable Attackers' HPCs:* Prior studies on SCAs detection have mostly focused on profiling both victim and attack applications to collect hardware performance counters data for detecting whether an attack occurs or not [8, 10, 42]. Nevertheless, a recent work [11] presents that attackers can craft malware to bypass the detectors. We believe this problem also challenges SCAs detection. What's more, SCAs can hide in benign applications, and their HPCs information is not available.

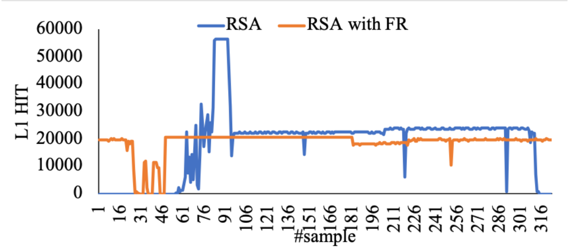


Figure 1: L1 HIT of RSA and RSA under Flush Reload attack

- *SCAs Design Principle:* Current SCAs intentionally cause influence on victim applications' cache or branch predictor by flushing/priming cache, mistraining branch predictors and then observe accessing time of the cache sets, which changes caching victims' data and microarchitectural behaviors of victim applications [41]. This also provides the opportunity of detecting SCAs by observing the alteration in microarchitectural behaviors. Furthermore, our experimental results as shown in Figure 1 indicate a clear difference between the behavior of VNA and VA. The HPC traces of L1 HIT for the tested victim application (RSA) under no attack (RSA) and under L3 Flush Reload (RSA with FR) are illustrated. It observes that the L1 HIT of VA shows a significantly different trend compared to that of VNA.

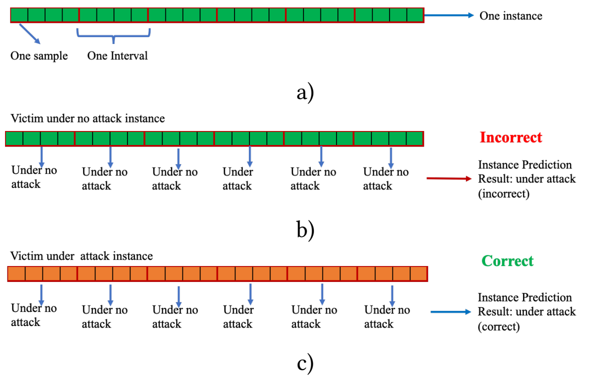

One instance
One sample
One Interval
a)
Victim under no attack instance
Incorrect
Under no attack
Under attack
Instance Prediction Result: under attack (incorrect)
b)
Victim under attack instance
Correct
Instance Prediction Result: under attack (correct)
c)

Figure 2: False Alarm problem: a) Concept; b) Victim under No Attack (VNA) condition; c) Victim under Attack (VA) condition

2.1.1 *False Alarm Challenge.* In Figure 2-(a), we show each run of a victim application which we refer to as an instance. For run-time SCAs detection, a certain window size is used to decide the number of samples of each interval. Each instance contains multiple intervals. Figure 2-(b) shows a VNA instance divided into multiple intervals. In such a case, even if only one interval is predicted as VA by the ML-based detection technique, the entire instance is classified incorrectly as VA. Figure 2-(c) illustrates an opposite case with a VA instance with five intervals classified as VNA and one interval classified as VA, resulting in the entire instance to be classified correctly as VA. To distinguish false positives of interval level and instance level, we employ **false alarm** and **missed alarm** terminology to represent false positive and false negative of instance level in the following sections.

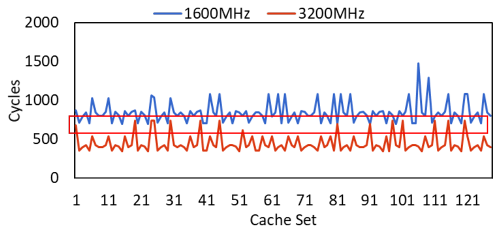


Figure 3: SCA probing time comparing 1600MHz and 3200MHz

2.2 System Level Parameter Adaptation: Frequency

According to a recent study [20], the threshold should be set depending on the platform and frequency used, e.g., 700 cycles for the server under 2900 MHz and 400 cycles for the desktop under 3200 MHz. To highlight the influence of frequency scaling, the Probing results is plotted under 1600 MHz and 3200 MHz in Figure 3. In Figure 3, X-axis represents the cache set, and Y-axis represents the accessing time in the unit of cycles. It observes that 700 cycles-per-accessing belongs to "long accessing time" (cache miss) under 3200MHz while it belongs to "short accessing time" (cache hit) under 1600MHz. It finds that the L3 cache hit threshold cannot separate the probing results highlighted in the rectangle region in the figure. Frequency randomization makes it difficult for the attacker to distinguish which cache set has been accessed.

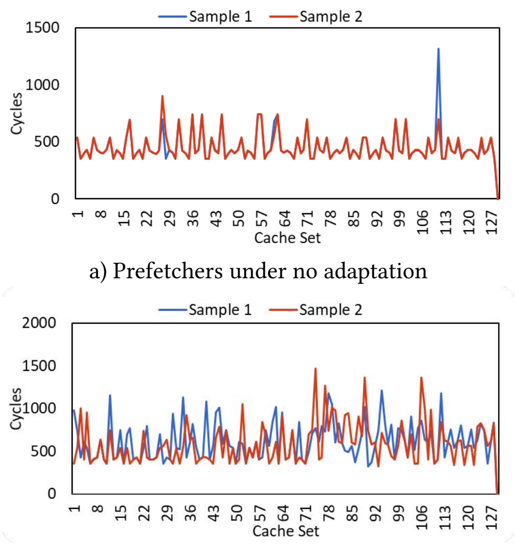


Figure 4: The impact of hardware prefectchers adaptation across two separate cache traces

2.3 Architecture Level Parameter Adaptation: Prefetchers

Prefetcher units are developed in modern microprocessors that are responsible for fetching the data/instructions predicted to be accessed in near future references [6, 14, 26, 27, 33]. As shown in Table 2, there are four prefetchers units implemented in various Intel processor architectures. The functionality of each prefetcher can be changed either through the BIOS setting or by directly writing a value to the register [1]. In this work, we perform the latter method to randomize the functionality of prefetchers during run-time. Moreover, implementation of cache-based SCAs like L3 Prime+Probe leverages a randomizing function for accessing memory to avoid being influenced by prefetchers. However, such methods are effective only when prefetchers are enabled or disabled constantly.Thus, here we examine the impact of randomly enabling prefectchers in SCAs' observations.

In Figure 4-(a) and Figure 4-(b), X-axis represents the cache set, and Y-axis represents the accessing time in the unit of cycles. Prime Probe is executed twice, and two cache accessing traces are collected with the same frequency setting at 3200 MHz. As shown in 4-(a) without prefetchers' randomization, in most cases, the two samples' traces overlap with each other, indicating that the attacker can deduce cache pattern of victim applications with multiple cache traces and remove noise. Figure 4-(b) shows the probing results of two samples under prefetchers' randomization (when prefetchers

Table 2: Different hardware prefetchers available in Intel computer architecture

Prefetcher	Description
DCU Hardware prefetcher	Fetches the next cache line into L1 data cache
DCU IP prefetcher	Uses sequential load history to determine whether to prefetch the next expected data into L1 cache from memory or L2
L2 hardware prefetcher	Fetches additional lines of code or data into the L2 cache
L2 adjacent cache line prefetcher	Fetches the cache line that comprises a cache line pair (128 bytes)

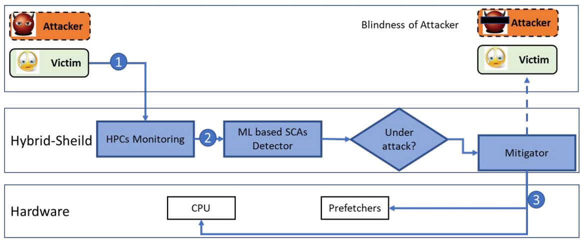


Figure 5: Overview of *d-Shield*

are randomly enabled or disabled). It notices that two traces vary significantly over the accessing time, and no specific cache access pattern can be identified. Thus, randomly enabled prefetchers can add more noises to cache access traces that attacker observes.

3 PROPOSED METHODOLOGY

3.1 Overview

To combat the cache-based SCAs, *Hybrid-Shield* includes a run-time detector that identifies the SCAs in the system. It then activates a mitigator module after receiving an attack alarm by randomizing a system (frequency) and adapting architecture level (prefetchers) parameters. In particular, each victim application is executed on a fixed core and HPCs are collected when the victim is running, as shown as Step 1 in Figure 5. Next, as depicted in Step 2 in Figure 5, the collected HPC data is fed to a trained machine learning based detector which decides if the victim application is under attack. If the detection result is "under attack", then as shown as Step 3 in Figure 5, the mitigator randomizes the settings of frequency and prefetchers to mitigate the impact of the detected SCA. Such randomization-based adjustment contaminates the cache traces and makes attacks effectively blind.

3.2 Threat Model

As for the threat model in our experiments, we consider multi-core computing environments that use the inclusive (L3 cache) and the non-inclusive cache memories (L1/L2 cache) on the Intel architecture, in which benign and malicious processes deploy shared libraries. It is assumed that the environment is Linux-based single OS environment, and both victim and attack applications reside in the same physical machine, either running on the same core or different processing core. The system could face side-channel attacks comprising of shared-memory based attacks (e.g. Flush+Reload), and also none shared-memory based attacks (e.g. Prime+Probe). It is further assumed that the potential SCAs might be hidden inside benign application or be crafted to emulate the benign application behavior.

3.3 Experimental Setup

In this work, all experiments are conducted on an Intel i5-3470 processor with Ubuntu 16.0.4 LST operating system with Linux kernel 4.13, 4 cores, 8GB DRAM, and a three-level cache system. The L1 and L2 caches are exclusively separated. L3 cache memory is inclusive and shared among all cores meaning that flushing out the data in the last level cache could remove the data in the L1 cache. The inclusiveness of L3 cache creates potential vulnerability surface for LLC attacks to be exploited.

3.4 ML based Run-time SCAs Detector

Figure 6 presents the proposed detector. It is comprised of different steps such as data collection, feature reduction, training phase, testing phase, and false alarm minimization method. First, for feature extraction, the "under no attack" and "under attack" HPC data is collected within a) isolated scenario, and b) non-isolated scenario. Then customized features are extracted and the data is used to train various classifiers. Next, the trained models are deployed in the testing phase and false alarm minimization technique further assists in reducing the false alarm rate.

Table 3: The collected HPC features and their ranking

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

3.4.1 Data Collection. In our case study, RSA and AES are used as victim applications, while Flush+Reload and Prime+Probe are used as attacks. Based on the behavior and functionality of studied SCAs, 16 HPC features are considered in this work for further analysis as listed in Table 3. These hardware performance counters data are collected at every 50 microseconds intervals. Each pair of a VNA and VA executes 50 times. Next, both VA and VNA HPC data are merged to create the final dataset. To validate each of the utilized ML classifiers, a standard 70%-30% non-biased dataset split for training and testing is followed.

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

Step 1. Features Vector Extraction and Representation. For proposed classification, time-series data is transformed from a time sub-sequence to a vector of features. In the first step of the transformation process, the raw data is received from the monitoring module. The time-series sub-sequences' properties is extracted, which includes statistics of distribution values (including max, min, StatAv, and sum). To effectively determine the most prominent features for run-time SCA detection, we deployed Greedy Forward Selection algorithm [4, 13] and we found that max, min, StatAv and sum contribute more to assisting in distinguishing the difference between "under no attack" and "under attack" conditions. Hence, the input for each transformation is $T = (t1, t2, ..., tm)$ where tm

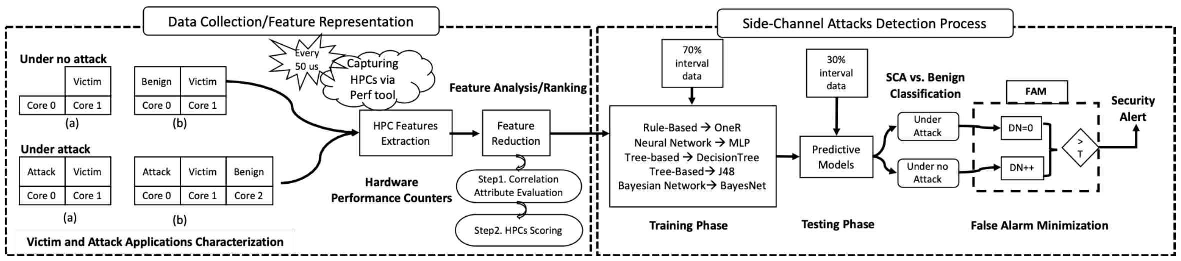


Figure 6: The overview of proposed data collection/feature representation and side-channel attacks detection process in *Hybrid-Shield*

is a vector of HPCs values. Also, the transformation outputs are a vector of actual HPC values, i.e., L1 HIT sum, L1 HIT max, etc.

Step 2. HPCs Feature Reduction. Given that there exists a limited number of HPCs available in modern microprocessors (only 4 HPCs on the tested Intel I5-3470) to be collected at once simultaneously, it is necessary to identify the most important HPCs for classifying the VA and VNA conditions. For HPCs reduction, we employ Correlation Attribute Evaluation ($CorrelationAttributeEval$ in Weka) with its default settings to calculate the Pearson correlation between attributes (HPC features) and class (VA and VNA conditions). Next, the sum score of each HPC feature (min, max, stdev, and sum in this work) will be calculated and HPCs will be ranked according to the sum score, as shown in Table 3.

3.4.3 ML Classifiers Implementation. For a thorough analysis of various types of ML classifiers, OneR, MLP (multilayer perceptron), DT (decision table), J48, and BayesNet algorithms are deployed as our final classification models. They are from different branches of ML (regression, neural network, decision tree, and rule-based techniques) and covering a diverse range of learning algorithms that are inclusive for modeling both linear and nonlinear problems. As mentioned, only four HPC features can be collected for most processors at once, due to a limited number of registers for storing them. Hence, reducing the number of HPCs required for ML models is vital to eliminate the need for multiple runs. For this purpose, various number of HPCs from 16 to 4 (16, 12, 8 and 4 selected based on the ranking in Table 3) are examined to evaluate the influence of reduced HPCs on classification accuracy and highlight the motivation of HPCs evaluation.

3.4.4 False Alarm Minimization (FAM). As mentioned in Section 2, previous run-time SCAs detection methods are biased to the under attack category that result in increasing the false alarm rate. Hence, to address this challenge, we propose False Alarm Minimization (FAM) technique in which we delay the "under attack" detection decision until receiving a certain number of continuous intervals, delay number (DN), before reporting as "under attack", while minimizing detection latency. Furthermore, this work demonstrates how delaying reporting "under attack" decision could ensure the false alarm rate being below a predefined threshold.

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

Table 4: Experimented randomization scenarios

Scenario	Frequency	Prefetchers
A	3200MHz	All Enabled
B1	1600~3200MHz	All prefetchers Enabled
B2	2600~3200MHz	All prefetchers Enabled
C1	3200MHz	DCU prefetcher Enabled
C2	3200MHz	DCU IP prefetcher Enabled
C3	3200MHz	L2 hardware prefetcher Enabled

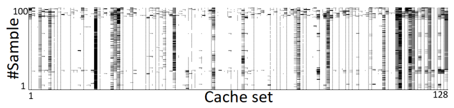
Table 5: A ~ C performance overhead normalized by A

Scenario	A	B1	B2	C1	C2	C3
Performance Overhead	1	1.43	1.23	1.2	1.29	1.20

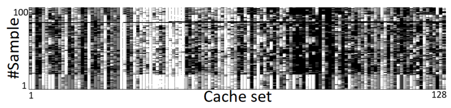
3.5 SCAs Mitigator

After receiving "under attack" alarm from the SCAs detector module, the SCAs mitigator is activated to protect the computer system from the SCAs threat in which the details are described below.

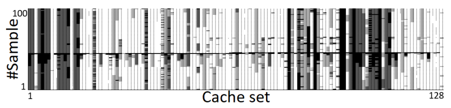
3.5.1 Hybrid SCAs Detection & Mitigation vs. Stand-alone Mitigation. To highlight the importance of randomizing both system level (frequency) and hardware level (prefetchers) concurrently, here we examine the performance overhead of stand-alone randomization-based mitigation. Taking RSA application under the Prime+Probe attack as an illustrative case study, six randomization scenarios are listed in Table 4. The corresponding cache patterns under different randomizing scenarios are shown in Figure 7. The black blocks represent the cache accessed by RSA. Hence, observing the clear black blocks patterns can help the attacker to extract users' information. Applying system and/or architecture level randomization increases the number of black blocks (accessed cache blocks), leading to substantial contamination of observed cache access patterns by the attacker. Also, as seen, randomization could mask the real victims' cache patterns. The performance overhead of various randomization scenarios is shown in Table 5. One can observe that scaling frequency and adapting prefetchers with various settings yield in different protection levels with various performance overheads. To cope with the high performance overhead, we propose to balance the need to eliminate side-channel information leakage and improve performance.



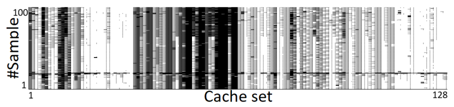
a) A: fixed frequency with enabled prefetchers



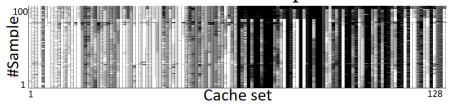
b) B1: randomized frequency within 1600MHz ~ 3200MHz and enabled prefetchers



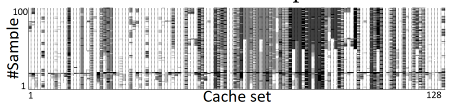
d) B2: randomized frequency within 2600MHz ~ 3200MHz and enabled prefetchers



e) C1: fixed 3200MHz frequency with DCU prefetcher adaptation



f) C2: fixed 3200MHz frequency with DCU IP prefetcher adaptation



g) C3: fixed 3200MHz frequency with L2 hardware prefetcher adaptation

Figure 7: RSA cache access heatmap under different randomization cases (A, B, C)

3.5.2 Randomization Evaluation Criteria. To determine comprehensive evaluation criteria for selecting the optimal randomization strategy, *Hybrid-Shield* accounts for both error rate and performance overhead evaluation metrics. The attack protection effectiveness represented by *ProtectionLevel* is calculated by normalizing the number of bits correctly recovered from cache traces under randomization with the number of bits recovered under no randomization. The higher the *ProtectionLevel* is, the more secure system is. In addition, the performance overhead referred to as *Overhead* indicates the total execution time under the applied mitigation strategy, which is normalized by the execution time under no mitigation strategy. As a result, we model the randomization evaluation metric in *Hybrid-Shield.* To examine the effectiveness and performance cost our randomization-based mitigation approach uses Equation 1 described below:

$$S = (\alpha * 1/ProtectionLevel) + (\beta * Overhead) \tag{1}$$

where S denotes the overall randomization score, α and β are the error rate and performance overhead, respectively. One can observe that the larger *ErrRate* provides a higher level of security and protection against SCAs, and the smaller *Overhead* indicates the less performance cost of the mitigation scenario. Hence, in our proposed randomization strategy described in Algorithm 1, we target to minimize the overall randomization score (S) to find the most efficient system and architecture level randomization scenario.

3.5.3 Randomization Strategy Selection. In this subsection, we present the details of the proposed randomization strategy selection in *Hybrid-Shield.* As mentioned in Section 2, there exist four prefetchers and frequency on our studied Intel processor system. We consider two sets of adjustments for each hardware prefetcher, including "1" as "deactivated" and "0" as "activated", while the frequency is ranged from 1600 MHz to 3200 MHz. Assuming that frequency value is $1600 + N * Step(n >= 0)$, then the number of possible frequency values is N. Hence, the number of prefetchers and frequency settings is $2 * 2 * 2 * 2 * N$. In this work, we set the initial value of Step as 400 MHz, corresponding to N=4. Moreover, the number of transitions from one setting to another setting is calculated: $2 * 2 * 2 * 2 * N = 16 * 4 = 64$. To further reduce the number of settings required to iterate, we propose a randomization strategy selection process shown in Algorithm **1** which helps in decreasing the number of the needed randomization settings search to 8~16.

As shown in Algorithm **1**, α, β, and ϵ are the input parameters to our proposed randomization algorithm where α and β help to change the weight of error rate (indicating protection level) and overhead, and ϵ is used to decide whether the gap between the two tested combinations are large enough to continue the search. In addition, S is the overall score of a randomization combination. If the gap is low enough, then the search stops; otherwise, the algorithm resumes the search till finding the most optimal combination. The value of frequency changes by 400 MHz ($Step$) each time. The values of 0-15 represent different prefetchers' combinations for hardware prefetchers, which could be increased or decremented each time by one. As seen in Algorithm **1**, the initial Strategy S_ini indicates that the frequency level is fixed at its highest value, and all the available hardware prefetchers are enabled. Next, the corresponding execution time (Exe_b) and recovered key (Key_b) are collected as listed in lines 4-5. Then, in lines 6-7 the algorithm generates a random strategy S_i and calculates the overall randomization score with α, β and S_i. From lines 7-25, the proposed randomization strategy proceeds with the same setting as S_i while increasing and decreasing the frequency by one Step (as long as the values of frequency are within range) resulting in new strategies of S_m and S_n as, respectively. As a result, the algorithm compares the score values of the two selected strategies and identifies the strategy with a larger score as a more efficient frequency level (S_f) to be used for the randomization. For prefetchers, in lines 26-41 the proposed algorithm follows a similar process as frequency selection and determines the optimal setting (S_p), which has a higher overall randomization score value. Next, if the minimum Score value among S_i, S_f and S_p is no ϵ smaller than S_i, then search stops; otherwise, it chooses the combination setting that generates the lowest Score value as a new S_i, and the search repeats from line 7.

4 EXPERIMENTAL RESULTS

4.1 SCAs Detection Evaluation

As described in Section 2, the high false alarm rate challenge significantly degrades the performance of SCAs detection. Hence, this section presents the performance evaluation results for both interval and instance level prediction.

4.1.1 Interval Level Evaluation: Detection Accuracy. Figure 8-(a) shows the SCAs detection accuracy with a varied number of HPCs for the proposed SCAs detector in *Hybrid-Shield* (customized features based classifiers) and existing works (traditional and time-series classifiers using different techniques) [8, 42]. One can observe that the time-series classifiers achieve lower accuracy despite utilizing more number of HPC features, i.e., the SCA detection accuracy

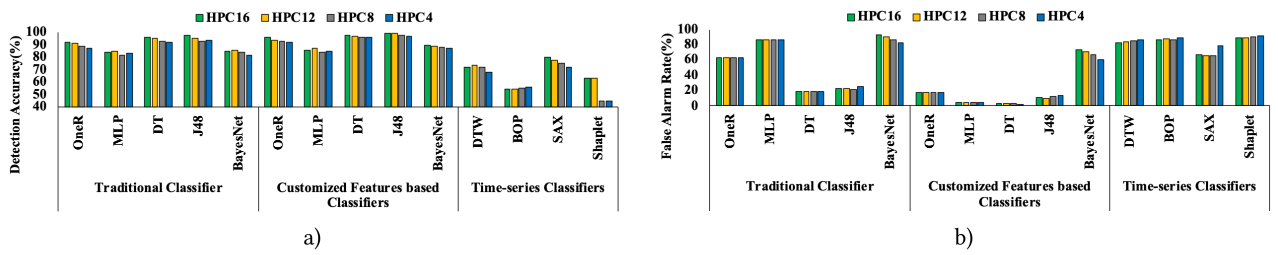


Figure 8: Comparison of a) detection accuracy and b) false alarm rate of proposed customized classifiers in *Hybrid-Shield* with traditional and time-series classifiers

Algorithm 1: Randomization in *Hybrid-Shield*

```
Input: $\alpha$,$\beta$,$\epsilon$ ($\alpha$ and $\beta$ decide the weight of error rate and overhead, and $\epsilon$ decides
        whether continue the search or not)
Result: Optimal Randomization Strategy
Step ←400MHz;
MinFrequency ←1600MHz;
MaxFrequency ←3200MHz;
S_i ←getRandomStrategy();
P_i←calculateScore($\alpha$, $\beta$, $S\_i$)
while TRUE do
    if S_i.frequency - Step >=MinFrequency then
        S_m ←S_i.frequency- step;
        Score_m ←calculateScore($\alpha$,$\beta$,S_m);
    else
        S_m ==NULL; Score_m =0
    end
    if S_i.frequency + Step <= MaxFrequency then
        S_n ←S_i.frequency+ step;
        Score_n ←calculateScore($\alpha$,$\beta$,S_n);
    else
        S_n ==NULL; Score_n =0
    end
    if Score_m >Score_n then
        S_f ←S_m;
        Score_f ←Score_m;
    else
        S_f ←S_n;
        Score_f ←Score_n;
    end
    if S_i.prefetchers - 1>= 0 then
        S_t ←S_i.prefetchers- 1;
        Score_t ←calculateScore($\alpha$,$\beta$,S_t);
    else
        S_t ==NULL; Score_t =0
    end
    if S_i.prefetchers + 1 <= 15 then
        S_j ←S_i.prefetchers+ 1;
        Score_j ←calculateScore($\alpha$,$\beta$,S_j);
    else
        S_j ==NULL; Score_j =0
    end
    if Score_t >Score_j then
        S_p ←S_t;
        Score_p ←Score_t;
    else
        S_p ←S_j;
        Score_p ←Score_j;
    end
    if Score_i-min(Score_f,Score_p,Score_i)<$\epsilon$ then
        return S_i
    end
    if Score_f ==min(Score_f,Score_p,Score_i) then
        S_i =S_f
    else
        S_i =S_p
    end
end
```

is < 70% on average. Hence, existing time-series classifiers are not the optimal solution for run-time SCAs detection. By comparison, the proposed and traditional classifiers achieve nearly 80% detection accuracy despite utilizing fewer HPCs, which makes them potential candidates for run-time SCAs detection.

Table 6: False alarm rate with different delayed interval

Classifiers / DN Value	OneR	MLP	DT	J48	BayesNet
1	20.4	4.7	2	13.6	61
2	16	0	0	1.7	27.8
4	3.6	0	0	0	1.8

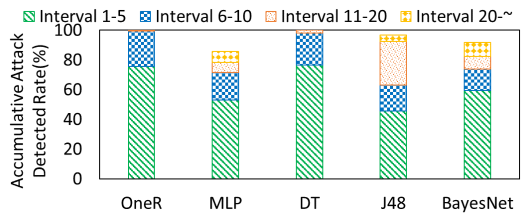


Figure 9: Attack detection speed and accuracy for customized classifiers with 4 HPCs when DN=2

4.1.2 *Instance Level Evaluation: False Alarm Rate.*

As discussed, despite high interval level prediction accuracy, one of the major challenges associated with efficient SCAs detection is false alarms. Figure 8-(b) depicts the false alarm rate of the proposed ML-based SCAs detection in *Hybrid-Shield* and existing techniques when utilizing a various number of HPCs for SCA detection. The false alarm rates of traditional classifiers based on SCAs detection is significantly high, 57% on an average across all ML techniques and HPC values. This is because traditional methods are biased to "under attack". However, the proposed *Hybrid-Shield* is able to predict "under attack" scenario with less bias and higher confidence. Taking MLP-based SCA detector as an example, the proposed customized classifier decreases false alarm rate from 87% (obtained when utilizing traditional classifier) to 4.7%, though the detection accuracies are similar. Furthermore, time-series classifiers have shown above 80% false alarm rate showing the inefficiency of these complex models for accurate run-time SCAs detection.

4.1.3 *SCAs Detection False Alarm and Latency of FAM.*

Table 6 shows false alarm rate results when delay intervals (DN) is set to 1, 2, and 4. Interesting for proposed customized OneR classifier in *Hybrid-Shield*, false alarm rate drops from 20.4% to 3.6% when increasing DN from 1 to 2. A similar trend can be observed for J48 classifier, dropping from 13.6% to 1.7% with DN increasing from 1 to 4. BayesNet SCA detector, however, requires DN=4 to achieve false alarm rate below 5%. The DN can be determined based on the equation provided in Section 3.4.4. In Figure 9, all "under attack" reports are delayed by two intervals. The X-axis indicates the five customized ML classifiers and Y-axis represents the percentage of all victim under attack instances reported as "under attack". "Interval 1-5" refers to the percentage of attacks reported in between interval 1 and interval 5, and other legends follow the same rule. The faster the reported attack, the less time the attack can remain in the system. For attack detection accuracy, it can be seen that only OneR and DT can achieve 100% attack detection accuracy. As for detection latency that corresponds to the classifiers' speed, it can be observed that OneR and DT perform the best among all experimented ML models reporting 80% under attack instances within 5 intervals and 100% within 20 intervals.

The proposed customized features based ML classifiers can achieve high detection accuracy and maintain a low false alarm rate. To further reduce the false alarm rate, the FAM technique is deployed and the "under attack" report is delayed by 2 intervals. The results indicate that the DT classifier can obtain 100% detection accuracy with 0% false alarm rate. In addition, nearly 80% of attacks can be detected within 5 intervals (2.5 milliseconds), 98% can be detected within 10 intervals (5 milliseconds), and only 2% are detected between 10-20 intervals (5-10 intervals).

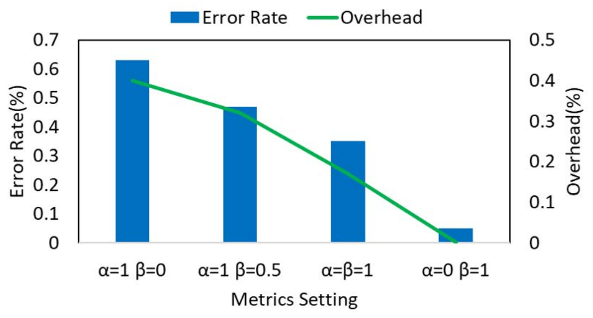


Figure 10: Flush+Reload Error Rate and RSA Overhead

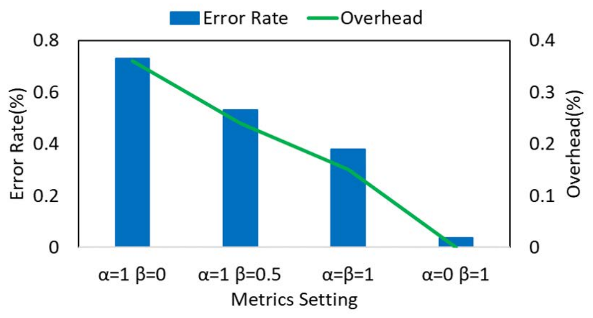


Figure 11: Prime+Probe Error Rate and RSA Overhead

4.2 SCAs Mitigation Evaluation

To evaluate the mitigation approach in *Hybrid-Shield*, Flush+Reload and Prime+Probe are used as two case studies. The two attacks belong to two different categories of side-channel attacks: shared-memory and no shared-memory, respectively. As mentioned before, α and β parameters are used to determine the weight of the protection level and performance overhead. We set α, β to various values such as 1, 0; 1, 0.5; 1, 1; 0, 1 as shown in Figure 10 and Figure 11 and ϵ is set as 0.05. One can observe that when α equals to 1 and β equals to 0, the error rate is the highest among the four settings. When α equals 0 and β equals 1, the performance is the most important evaluation metric and no randomization is adopted to protect victim applications. The overhead in both Figure 10 and Figure 11 is calculated by normalizing the execution time by the time when α equals 0 and β equals 1.

For both Flush+Reload and Prime+Probe, it can be observed that the error rate for attacks can be as high as 52% and 73% when performance overhead is not taken into consideration (α=1, β=0). However, under this mode, the attacks can be significantly undermined. By increasing the weight of the performance overhead, both the error rate and performance overhead decreases. When the error rate and performance overhead are given the same level of importance (α=1 β=1), the SCAs mitigator can result in 35% error rate with 17% performance overhead for Flush+Reload, and 38% error rate with 15% performance overhead for Prime+Probe. Hence, it can be concluded that the proposed SCAs mitigation methodology can effectively pollute victims' cache patterns and change the protection level and performance overhead by the setting.

5 CONCLUSION

Microarchitectural Side-Channel Attacks (SCAs) have posed serious threats to the security of modern computing systems. Prior efforts have focused on identifying these security vulnerabilities using low-level microarchitectural features, and designing countermeasures for them. In this work, we have identified some important challenges associated with effective run-time SCAs detection and mitigation that have been neglected in prior studies. To this aim, we proposed *Hybrid-Shield*, an accurate and efficient cross-layer countermeasure solution for run-time detection and mitigation of cache-based side-channel attacks. For the detection stage, microarchitectural information of victim application under attack and under no attack conditions are collected for training various types of Machine Learning (ML) classifiers. The ML-based detectors are then deployed to facilitate accurate run-time detection of SCAs using only the HPC features of victim application with high detection accuracy and low instance level false alarm rate. For the mitigation stage, *Hybrid-Shield* offers a lightweight system and architecture level randomization technique to efficiently mitigate the impact of cache-based side-channel attacks with no hardware redesign overhead. By carefully adapting the processor frequency and prefetchers configurations and adding a proper level of noise to the attackers' cache observations, *Hybrid-Shield* protects the critical information from being leaked, enhancing the security of the computer system. The experimental results showed that the proposed hybrid detection and mitigation countermeasure against timing-based side-channel attacks achieves up to 100% detection accuracy with 0% false alarm rate. After the capturing SCAs, the mitigator component is activated that outperforms the state-of-the-art SCAs mitigation solutions achieving significantly lower performance overhead, reducing the performance loss from 32.66% down to 15%.

6 ACKNOWLEDGMENT

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

REFERENCES

[1] Disclosure of h/w prefetcher control on some intel processors. In *https://software.intel.com/en-us/articles/disclosure-of-hw-prefetcher-control-on-some-intel-processors*.

[2] Al-Dujaili, A., Huang, A., Hemberg, E., and O'Reilly, U.-M. Adversarial deep learning for robust detection of binary encoded malware. In *2018 IEEE Security and Privacy Workshops (SPW)* (2018), IEEE, pp. 76–82.

[3] Briongos, S., et al. Cacheshield: Detecting cache attacks through self-observation. In *ACM Conference on Data and Application Security and Privacy* (2018), ACM.

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

[5] Chen, S., Zhang, X., Reiter, M. K., and Zhang, Y. 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* (2017), ACM, pp. 7–18.

[6] Chen, T.-F., and Baer, J.-L. Effective hardware-based data prefetching for high-performance processors. *IEEE transactions on computers 44*, 5 (1995), 609–623.

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

[8] Chiappetta, M., Savas, E., and Yilmaz, C. Real time detection of cache-based side-channel attacks using hardware performance counters. *Applied Soft Computing 49* (2016), 1162–1174.

[9] Cho, J., Kim, T., Kim, S., Im, M., Kim, T., and Shin, Y. Real-time detection for cache side channel attack using performance counter monitor. *Applied Sciences 10*, 3 (2020), 984.

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

[11] Dinakarrao, S. M. P., Amberkar, S., Bhat, S., Dhavlle, A., Sayadi, H., Sasan, A., Homayoun, H., and Rafatirad, S. Adversarial attack on microarchitectural events based malware detectors. In *Proceedings of the 56th Annual Design Automation Conference 2019* (2019), pp. 1–6.

[12] Domnitser, L., and et.al. Non-monopolizable caches: Low-complexity mitigation of cache side channel attacks. *ACM TACO* (2012).

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

[14] González, A., and et.al. A data cache with multiple caching strategies tuned to different types of locality. In *International Conference on Supercomputing* (1995).

[15] Grosse, K., Papernot, N., Manoharan, P., Backes, M., and McDaniel, P. Adversarial examples for malware detection. In *European Symposium on Research in Computer Security* (2017), Springer, pp. 62–79.

[16] Gruss, D., and et.al. Flush+ flush: a fast and stealthy cache attack. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment* (2016).

[17] Kolosnjaji, B., Demontis, A., Biggio, B., Maiorca, D., Giacinto, G., Eckert, C., and Roli, F. Adversarial malware binaries: Evading deep learning for malware detection in executables. In *2018 26th European Signal Processing Conference (EUSIPCO)* (2018), IEEE, pp. 533–537.

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

[19] Liu, F., and et. al. Catalyst: Defeating last-level cache side channel attacks in cloud computing. In *2016 HPCA*.

[20] Liu, F., and et.al. Last-level cache side-channel attacks are practical. In *SP* (2015), IEEE, pp. 605–622.

[21] Liu, F., and Lee, R. B. Random fill cache architecture. In *47th Annual IEEE/ACM International Symposium on Microarchitecture* (2014).

[22] Liu, F., Wu, H., Mai, K., and Lee, R. B. Newcache: Secure cache architecture thwarting cache side-channel attacks. *IEEE Micro 36*, 5 (2016), 8–16.

[23] Mi, Z., and et.al. Cpu elasticity to mitigate cross-vm runtime monitoring. *IEEE Transactions on Dependable and Secure Computing* (2018).

[24] Mukhtar, M. A., Mushtaq, M., Bhatti, M. K., Lapotre, V., and Gogniat, G. Flush+ prefetch: A countermeasure against access-driven cache-based side-channel attacks. *Journal of Systems Architecture 104* (2020), 101698.

[25] Mushtaq, M., et al. 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* (2018), ACM, p. 1.

[26] Nesbit, K. J., and et.al. Ac/dc: An adaptive data cache prefetcher. In *Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques*.

[27] Nesbit, K. J., and Smith, J. E. Data cache prefetching using a global history buffer. In *10th HPCA'04*.

[28] Osvik, D. A., Shamir, A., and Tromer, E. Cache attacks and countermeasures: the case of aes. In *Cryptographers' Track at the RSA Conference* (2006), Springer, pp. 1–20.

[29] Page, D. Partitioned cache architecture as a ėide-channel defence mechanism.

[30] Prada, I., Igual, F. D., and Olcoz, K. Detecting time-fragmented cache attacks against aes using performance monitoring counters. *arXiv preprint arXiv:1904.11268* (2019).

[31] Sayadi, H., et al. Recent advancements in microarchitectural security: Review of machine learning countermeasures. In *63rd IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)* (2020), IEEE.

[32] Sayadi, H., Patel, N., PD, S. M., Sasan, A., Rafatirad, S., and Homayoun, H. Ensemble learning for effective run-time hardware-based malware detection: A comprehensive analysis and classification. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)* (2018), IEEE, pp. 1–6.

[33] Vanderwiel, S. P., and Lilja, D. J. Data prefetch mechanisms. *ACM Computing Surveys (CSUR) 32*, 2 (2000), 174–199.

[34] Wang, H., et al. Comprehensive evaluation of machine learning countermeasures for detecting microarchitectural side-channel attacks. In *Great Lakes Symposium on VLSI (GLSVLSI)* (2020), ACM.

[35] Wang, H., Sayadi, H., Mohsenin, T., Zhao, L., Sasan, A., Rafatirad, S., and Homayoun, H. Mitigating cache-based side-channel attacks through randomization: A comprehensive system and architecture level analysis. In *DATE'20* (2020), IEEE.

[36] Wang, H., Sayadi, H., Rafatirad, S., Sasan, A., and Homayoun, H. Scarf: Detecting side-channel attacks at real-time using low-level hardware features. In *2020 IEEE 26th International Symposium on On-Line Testing and Robust System Design (IOLTS)* (2020), IEEE, pp. 1–6.

[37] Wang, Z., and Lee, R. B. A novel cache architecture with enhanced performance and security. In *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*.

[38] Wang, Z., and Lee, R. B. New cache designs for thwarting software cache-based side channel attacks. In *ACM SIGARCH Computer Architecture News* (2007).

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

[40] Yarom, Y., and Falkner, K. Flush+ reload: A high resolution, low noise, l3 cache side-channel attack. In *USENIX Security Symposium* (2014), vol. 1, pp. 22–25.

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

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