Mitigating Cache-Based Side-Channel Attacks through Randomization: A Comprehensive System and Architecture Level Analysis

Han Wang[1], Hossein Sayadi[2], Tinoosh Mohsenin[3], Liang Zhao[4],
Avesta Sasan[4], Setareh Rafatirad[4], and Houman Homayoun[1]
[1]University of California, Davis, CA, USA
[2] California State University, Long Beach, CA, USA
[3] University of Maryland, Baltimore County, MA, USA
[4] George Mason University, Fairfax, VA, USA
[1]{hjlwang,hhomayoun}@ucdavis.edu, [2]{hossein.sayadi}@csulb.edu,
[3]{tinoosh}@umbc.edu,[4]{lzhao9,asasan,srafatir}@gmu.edu

***Abstract*—Cache hierarchy was designed to allow CPU cores to process instructions faster by bridging the significant latency gap between the main memory and processor. In addition, various cache replacement algorithms are proposed to predict future data and instructions to boost the performance of the computer systems. However, recently proposed cache-based Side-Channel Attacks (SCAs) have shown to effectively exploiting such a hierarchical cache design. The cache-based SCAs are exploiting the hardware vulnerabilities to steal secret information from users by observing cache access patterns of cryptographic applications and thus are emerging as a serious threat to the security of the computer systems. Prior works on mitigating the cache-based SCAs have mainly focused on cache partitioning techniques and/or randomization of mapping between main memory. However, such solutions though effective, require modification in the processor hardware which increases the complexity of architecture design and are not applicable to current as well as legacy architectures. In response, this paper proposes a lightweight system and architecture level randomization technique to effectively mitigate the impact of side-channel attacks on last-level caches with no hardware redesign overhead for current as well as legacy architectures. To this aim, by carefully adapting the processor frequency and prefetchers operation and adding proper level of noise to the attackers' cache observations we attempt to protect the critical information from being leaked. The experimental results indicate that the concurrent randomization of frequency and prefetchers can significantly prevent cache-based side-channel attacks with no need for a new cache design. In addition, the proposed randomization and adaptation methodology outperforms the stat-of-the-art solutions in terms of the performance and execution time by reducing the performance overhead from 32.66% to nearly 20%.**

***Index Terms*—SCA mitigation, frequency scaling, prefetcher adaptation, randomization, Prime+Probe**

I. Introduction

In the last few decades, the complexity of computing systems has extensively increased to support different functionalities and meet the performance demands of emerging applications. Despite the provided performance benefits, the security of such systems has been exploited by numerous attacks [17], [16]. Side-Channel Attacks (SCAs) are a class of attacks that primarily exploit the computer systems vulnerabilities to infer sensitive information and confidential data by observing *side-channel information* [2], [22].

Timing-based cache SCAs [21], [10], [8] can be launched by the attacker remotely (e.g. attacks can even occur in cloud environments). Such attacks exploit the accessing time gap between the on-chip caches and main memory, and collect cache hit/miss traces based on various accessing times. Hence, the attacks can infer sensitive information according to cache traces and the knowledge captured from the cryptographic algorithm. There exists a number of cache-based SCAs proposed in prior studies [21], [10], [5], [14] causing a substantial threat to the security of modern computer systems. Due to the invisibility, feasibility, and capability to expose and extract the secret keys in the cache-based SCAs, there is an urgent need to address the security risks posed by such attacks in present computer systems as well as legacy systems [5], [21].

Prior works on cache-based side-channel attacks mitigation can be categorized into two main designing principles including cache partitioning [9], and randomization-based techniques [20]. Cache partitioning methods have been proposed to isolate cache usage between different programs to prevent the attackers from observing the victims data access patterns and stealing the confidential information stored in the cache memory [15], [20], [9]. In general, the cache partitioning techniques divide the cache memory into different zones for different application processes statically or dynamically. As a result, attack applications do not have access to observing cache access of users' applications. Another approach that was proposed to mitigate the impact of SCAs is based on randomization techniques [19], [11] in which they attempt to randomize the memory-to-cache mappings. These methods were mostly proposed in the architecture community as a solution to protect future architectures, and they are not applicable to current as well as legacy architectures since they require hardware redesign efforts.

In response, this work proposes a comprehensive system and architecture level randomization methodology to efficiently mitigate the impact of side-channel attacks on last-

978-3-9819263-4-7/DATE20/©2020 EDAA

level caches eliminating the need to modify the cache memory architecture. To this aim, by carefully adapting the processor frequency and prefetchers operation, we attempt to change the attackers' observation from victim application's cache access pattern and protect the critical information from being leaked. Our proposed randomization methodology shows that scaling frequency can change the attackers observations by changing the accessing time while the L3 cache hit threshold used to distinguish cache hit and miss is fixed. Prefetchers adaptation is also shown to be effective by adding random noises to victim cache traces. The proposed randomization-based approach indicate the possibility of hiding victim cache trace and protecting victims' information from being leaked. In order to show the effectiveness of our proposed side-channel attacks mitigation methodology, L3 Prime+Probe [10] is employed as a case study, since this attack does not require shared processor core or memory posing a greater security threat compared to other existing cache-based SCAs.

II. Background and Relevant works

In this section, we describe the background and related studies on SCAs mitigation techniques.

A. Prefetcher

To further reduce the large access latency of main memory, prefetcher units are developed in modern microprocessors that are responsible for fetching the data (as well as instructions) that will be more likely accessed in near future and bringing them from off-chip main memory to the cache [3], [4], [18], [13]. As shown in Table I, there exist four prefetchers units in various Intel processor architectures such Nehalem, Westmere, Sandy Bridge, Ivy Bridge, Haswell, and Broadwell. On each core, there is a Model Specific Register (MSR) with the address of 0x1A4 that can be used to control the 4 prefetchers [1]. Bits 0-3 of the MSR are used to control functionality of the prefetchers. When the corresponding bit is set to 1, the prefetcher is disabled; otherwise, it is enabled. The value of the register can be changed either through the BIOS setting or directly writing a value to the register. In this work, we perform the latter method and change the functionality of prefetchers randomly during the execution of victim applications.

B. Prime+Probe Attack

The Prime+Probe attack contains two steps: 1) "prime": evicting cache sets that consist of victim's data with potential conflicting memory blocks; 2) "probe": accessing data of the memory blocks and measuring the access time. Compared to L1 Prime+Probe, L3 last level cache Prime+Probe attack is more challenging due to the fact that L3 cache has a much larger size (6MB in this work) with higher latency compared to L1 cache, which makes the probing phase a more difficult and time-consuming process. Furthermore, current Intel processors divide the last level cache into different partitions each connected to different cores, hardening the recovery of mapping address by the attacker. Hence, L3 Prime+Probe attack firstly attempts to find potential conflicting cache sets to narrow down the scope of Probing step and achieve high attack resolution, which is a critical step for successful attacks. For the "Probing" step, it randomizes eviction sets to eliminate the influence of prefetchers and re-access memory lines. Since in this work we are focusing on Prime+Probe attack as case study, below we briefly describe the important steps for finding potential conflicting cache sets used in [10] which proposed and implemented L3 Prime+Probe.

- **Step 1.** Build a Large Page. Large page size can eliminate the need of address translation. Here, the potential conflicting memory lines are setup.
- **Step 2.** Expand. Iteratively add lines to the subset initialized as an empty subset as long as there is no self-eviction. Self-eviction is detected by priming a potential new member, accessing the current subset and timing another access to the potential new member.
- **Step 3.** Contract. Iteratively remove lines from the subset checking for self-eviction.
- **Step 4.** Collect. Scan original set, looking for members that conflict with the contracted subset.
- **Step 5.** Repeat until the original set is (almost) empty.

C. Relevant Works on SCA Mitigation

In this section, we present the latest studies on side-channel attacks mitigation and securing the computer systems against information leakage. Several recent works have proposed to modify cache hierarchy or cache memory architecture to mitigate SCAs. Cache partitioning techniques [15], [9] are proposed to mitigate cache-based side-channel attacks by statically or dynamically partitioning cache memory for each application process, thereby SCAs are not able to observe "side-channel information" of victim applications. Another approach employs access randomization [20], [19], [11] which primarily randomizes cache interference, remaps cache indices, or replaces demand fetch with random cache fill to eliminate security vulnerabilities in the hardware architecture. However, such works require new design of cache and cache memory translation architectures which poses extra design costs and can not be applied to legacy systems.

A recent work [12] on SCA mitigation has proposed to scale frequency to hide the victim's cache trace. In particular, it presents a general and elastic protection scheme against SCAs in the cloud environment. It requires purchasing a higher clock rate for protected VM and it lowers the frequency of suspected malicious VM when security-critical operations happen. While this method is effective in addressing the issue of Flush+Reload mitigation, it introduces significant performance overhead of more than 33.7% to the system. In addition, the authors limit their study only on the frequency randomization to pollute cache trace and also their method requires the knowledge of the processing core where victim applications and malicious applications reside, which is difficult to obtain in local environment.

III. Proposed Methodology

In this section, we present the details of the proposed randomization methodology for mitigating the cache-based side-channel attacks by adapting the frequency and prefetchers in modern computer systems as well as legacy systems.

TABLE I: 4 Prefetchers in Intel Computer Architecture

Prefetcher	Description
DCU Hardware prefetcher	which 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)

TABLE II: Architectural configurations

Processor	Intel I5-3470 CPU, single socket-4 cores
Frequency	1.6-3.2GHz
L1i Cache	32KB
L1d Cache	32KB
L2 Cache	256
L3 Cache	6144KB
Memory Capacity	8GB DDR3

A. 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. In Intel I5-3470 on-chip cache memory subsystem, while L1 and L2 caches are exclusively separated, and L3 cache memory is inclusive and shared among all cores meaning that flushing out the data in the last level cache could also remove the data in L1. The inclusiveness of L3 cache provides a potential vulnerability for LLC attacks to be exploited. The details of cache hierarchy are given in Table II.

TABLE III: Experiment Scenarios

Scenario	Frequency	Prefetchers
A	3200MHz	All Enabled
B1	1600∼3200MHz	All prefetchers Enabled
B2	2200∼3200MHz	All prefetchers Enabled
B3	2600∼3200MHz	All prefetchers Enabled
C1	3200MHz	DCU prefetcher Enabled
C2	3200MHz	DCU IP prefetcher Enabled
C3	3200MHz	L2 hardware prefetcher Enabled
C4	3200MHz	L2 adjacent cache line prefetcher Enabled
D	2600∼3200MHz	DCU IP prefetcher Enabled

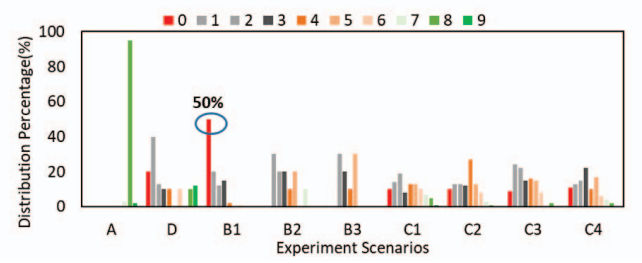


Fig. 1: Group size of potential eviction sets (ranging from 0∼9) comparisons among A, B, C and D experiment scenarios

B. Randomization Scenarios

Here, we introduce the studied scenarios for exploring the impact of randomizing prefetchers as an architecture-level and processor frequency as a system-level parameter on securing the computer system against SCAs. By accounting for various randomization combinations, we attempt to conduct L3 Prime+Probe as a case study to collect observable cache patterns in which the attacker tries to obtain and extract sensitive information. In this work, the cache patterns will be employed to analyze the effectiveness of the proposed methodology. In addition, all experimented settings are listed in Table III.

Scenario A: In order to obtain cache pattern information with less noise, Prime+Probe fixes frequency and avoids fluctuation in accessing cycles resulted from different frequencies. Hence, in scenario A we fix frequency to 3200MHz and enable all prefetchers. Scenario A is used as a comparison baseline with other randomization cases.

Scenario B1∼B3: Scaling frequency can change accessing time used by attackers to determine whether cache sets are accessed by the victim or not. Since scaling frequency changes applications' execution time (performance), we choose three ranges to evaluate the influence of scaling frequency values on victim's cache pattern and performance including 1600MHz ∼ 3200MHz, 2200MHz ∼ 3200MHz, and 2600MHz ∼ 3200MHz as listed in Table III. It is notable that the larger the range is, the slower the performance becomes, since applications will be executed under lower frequency.

Scenario C1-C4: As mentioned, the Intel I5 cores have 4 prefetchers that can be enabled or disabled by writing the value to the memory address of 0x1A4 [7]. In these experiments, each of the four studied scenarios only enables one prefetcher for a random interval. Under such scenarios, the prefetched data will pollute attacker observations since some data will be evicted due to the prefetching process. To this aim, four scenarios are devised to evaluate the effectiveness of hiding the victim's cache pattern for different prefetchers.

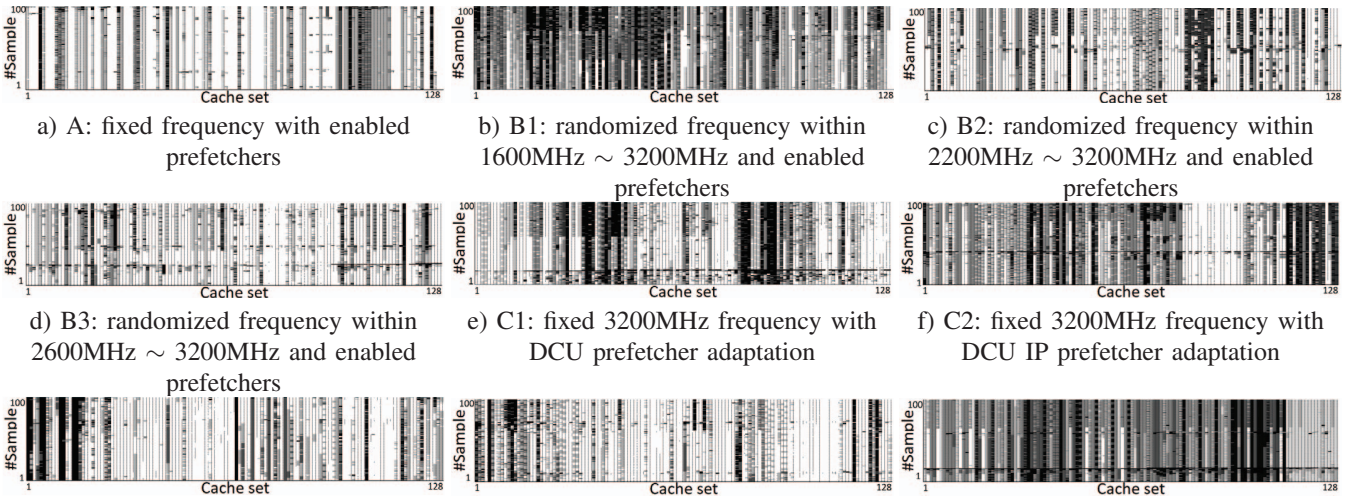
Scenario D: In this scenario which is considered as the combined scenario for adapting frequency and prefetchers, the least frequency scaling range and the most effective prefetcher to hide victims' cache pattern are tuned concurrently. The prefetcher size is chosen based on the result of C1 ∼ C4.

C. Experimental Methodology

As mentioned before, last level cache Prime+Probe attack is used as a case study to evaluate the effectiveness of our proposed approach. Due to the behavior of L3 Prime+Probe attack, here we evaluate the randomization influence based on two important factors including the size of eviction sets and probing results. To this aim, two different experimental methodologies are adopted and detailed below:

1) Eviction Set-based Analysis: For thoroughly analyzing the eviction sets results, we have considered three different applications running at the same time including a) victim; b) attack, where the potential eviction sets are built first and then probing the potential eviction sets takes place; and c) applying various randomization scenarios for SCA mitigation (discussed in Section III-B). As a result, eviction sets building process will be under randomization influence and the group size of eviction sets can be successfully collected.

2) Probing-based Analysis: As mentioned earlier, influence on eviction sets needs to be removed to obtain the impact of probing results. For this purpose, in our comprehensive



a) A: fixed frequency with enabled prefetchers

b) B1: randomized frequency within 1600MHz $\sim$ 3200MHz and enabled prefetchers

c) B2: randomized frequency within 2200MHz $\sim$ 3200MHz and enabled prefetchers

d) B3: randomized frequency within 2600MHz $\sim$ 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

h) C4: fixed 3200MHz frequency with L2 adjacent prefetcher adaptation

i) D: randomized frequency within 2600MHz $\sim$ 3200MHz with C2

Fig. 2: AES cache access heatmap under different randomization cases (A, B, C and D)

analysis, various randomization scenarios begin after building the eviction sets step, and victim and attack applications are executed concurrently. As a result, once the attacker prepares the eviction sets, probing and randomization scenarios start concurrently.

IV. Experimental Results and Evaluation

In this section, we present the experimental results and evaluation analysis of the proposed randomization-based SCA mitigation methodology. As described in the proposed methodology, we extract eviction set size and probing phase results of scenario A, and compare it with randomization cases (B, C, and D) to effectively analyze the level of noise added in attackers' observed information.

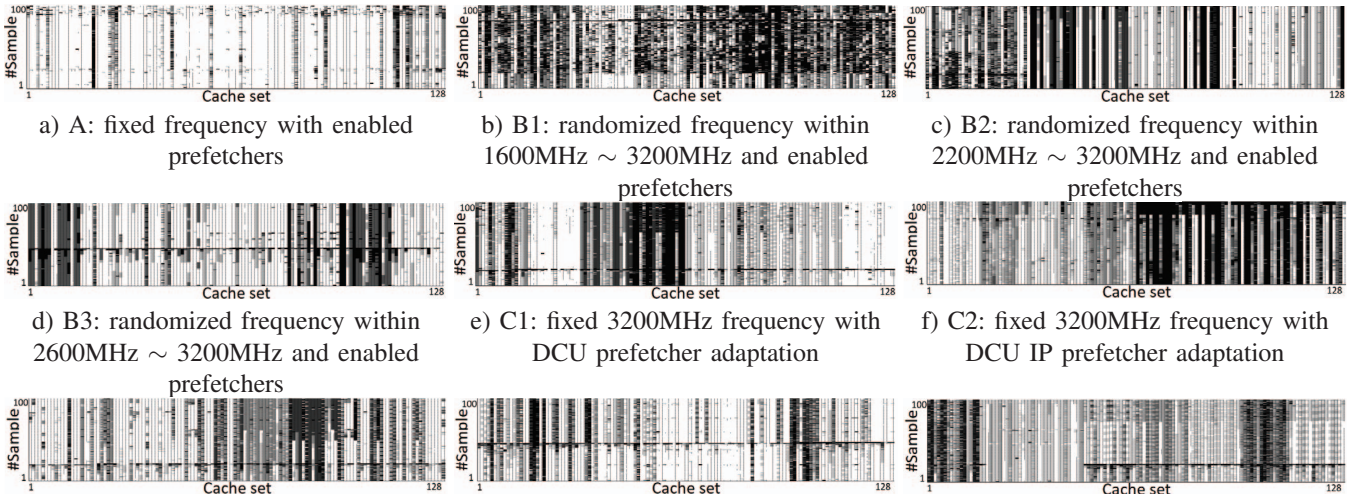
A. Eviction Sets-based Randomization

AES and RSA applications under L3 Prime+Probe are executed under all nine scenarios and different group sizes of eviction cache sets are collected. In order to effectively avoid from possible noises in the results, the attack is executed one hundred times. Figure 1, depicts the eviction sets results ranging from 0 to 9 across various randomization scenarios. As shown, X-axis represents different scenarios and Y-axis represents the number of eviction sets group size.

1) Frequency Randomization Analysis: For scenario B1 under frequency randomization, the result depicted in Figure 1 shows that there is a 50% possibility that the attacker builds 0 eviction set, indicating that the attacker is not able to find eviction sets and the attack will not be successful. Furthermore, a large percentage (above 40%) of samples group size is below or equal to 3, indicating a high possibility of missing the real conflict cache set. However, reducing scaling range (B2 and B3) decreases the effectiveness of causing the failure of building eviction sets. This fact indicates that scaling frequency with large range can confuse cache hit/miss during the time that attacker application is building eviction sets. For all three frequency scaling scenarios, eviction sets are all lower than scenario A. There exist two main reasons for this phenomenon. First, scaling frequency withholds the execution of attackers programs making some of the attackers' cache sets to be evicted due to frequency scaling. Secondly, scaling frequency changes the execution performance of both victims and attackers instructions, which results in the wrong conflict eviction set found in Step 2, Expand. As a result, once the attacker expands the cache sets by re-accessing the cache sets to find the conflicts, this process can misguide the attacker's observation in identifying the potential victim sets to comprise the security of the system.

2) Prefetchers Adaptation Analysis: Similarly, Figure 1 compares eviction group size of under C1$\sim$C4 scenarios. As can be seen, nearly 10% of samples are 0 meaning that no potential eviction set is found and the attack can not proceed. It can also be observed that the group sizes of 100 samples are evenly distributed across 1$\sim$7 eviction sizes. This will result in two observations that affect the success of attacks in leaking information. First, the attacker can miss the actual conflict cache lines. Second, the attacker is not able to collect the aligned cache traces each time while L3 Prime+Probe requires 300$\sim$1000 sample traces [10]. Furthermore, it is notable that different prefetchers components have a relatively similar effect on evictions sets building results. Our comprehensive analysis shows that compared with frequency scaling, adapting prefetchers is less effective in preventing attackers to identify potential conflicts for mitigating the attacks. This is because prefetchers can only change instructions' execution of victim and attack applications running on the system when requested data is being fetched in advance. That being said, prefetchers adaptation still shows high effectiveness for causing a disturbance in attackers observation.

3) Analysis of Concurrent Adaption of Frequency and Prefetchers: Scenario D contains both frequency scaling and prefetchers adaptation. As shown in figure 1, the distribution shows a more similar trend to the one observed in scenario



a) A: fixed frequency with enabled prefetchers

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

c) B2: randomized frequency within 2200MHz ∼ 3200MHz and enabled prefetchers

d) B3: 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

h) C4: fixed 3200MHz frequency with L2 adjacent prefetcher adaptation

i) D: randomized frequency within 2600MHz ∼ 3200MHz with C2

Fig. 3: RSA cache access heatmap under different randomization cases (A, B, C and D)

C in which around 20% of the SCA attacks are failed due to the failure of building potential conflict sets as described in Section II-B.

B. Probing Results

According to the behavior of L3 Prime+Probe attack, we probe the potential conflicting cache set (eviction cache set) and use heatmap analysis to examine the victim applications' cache accessing pattern. RSA and AES are used as victim applications. To show the probing results which can not be separated with the threshold, the heatmaps are plotted with 400 cycles suggested in [10] for the hit/miss threshold. As shown in Figure 2 and Figure 3, black blocks represent cache misses, meaning that victim accessed the cache set; white blocks represent cache hit that corresponds to the case in which the victim did not access the cache set. Hence, the attackers need to obtain more clear black blocks to find out the cache access pattern of the victim application for leaking information.

1) Frequency Randomization Analysis: As shown in the results, as compared to scenario A, both Figure 2-b), c) and d) and Figure 3-b), c) and d) show that the trace of victim has been significantly polluted with cache misses resulted by the frequency randomization. Furthermore, it can be observed that the larger the gap of frequency scaling range is, the higher noise the victims' cache trace has. Comparing B1, B2 and B3 in both figures depicts that scaling frequency from 1600MHz to 3200MHz can obtain higher noise and hide victim cache access pattern more efficiently. Such a phenomenon is because scaling frequency can change cache accessing time and mislead attackers leading them to identify cache hit/miss incorrectly. In addition, by increasing the gap, cache hit under low frequency and cache miss under high frequency are more likely to overlap, making cache traces contaminated. In all three frequency randomization scenarios (B1, B2, and B3) for both AES and RSA applications, no victim cache access pattern can be found, indicating that the attackers are not able to infer victim's secret information.

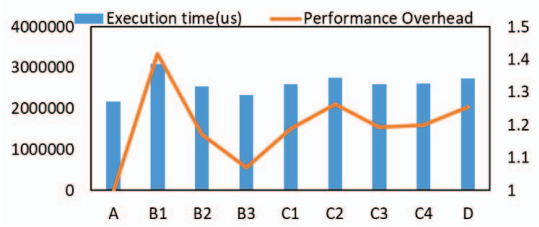


Fig. 4: A ∼ D execution time and performance overhead where A ∼ D is normalized by A

2) Prefetchers Adaption Analysis: Both Figure 2 and Figure 3 illustrate cache traces under different prefetcher settings. Generally, our comprehensive analysis across various configurations and scenarios indicates that randomly enabling/disabling different prefetchers (C1∼C4) results in less contamination of cache traces compared to frequency scaling. This is due to the fact that prefetcher units can only evict cache sets while scaling frequency influences all cache sets accessing time. Among all prefetchers, it can be found that DCU IP prefetchers adaptation (C2) is the most effective one compared to the other scenarios. Hence, in our optimal case study (Scenario D), DCU IP prefetcher is chosen as the target prefetcher unit to be tuned concurrently with frequency.

3) Concurrent Adaption of Frequency and Prefetchers: As can be observed from the results, for both AES and RSA benchmarks, cache traces resulted from optimal scenario of D in which both frequency and prefetcher units are selected at their optimal values, the concurrent adaptation shows more efficiency than the conditions of scaling frequency or adapting DCU IP prefetcher solely. The pollution extent of D is similar to the B1 scenario, indicating scaling frequency with a smaller range by adding prefetcher adaption.

C. Performance Overhead Analysis

In this Section, we choose MiBench [6] benchmark suites to evaluate the performance overhead of applications caused by randomizing frequency and prefetchers in terms of application execution time. In order to eliminate the influence of random noise on execution time caused by system scheduling, etc., each application from the benchmark under the eight scenarios is executed 100 times to avoid the interference of random noises. Figure 4 shows the arithmetic average execution time and performance overhead where A to D settings are normalized by the execution time of A.

From Figure 4, it can be observed that scaling frequency with largest range (Scenario B1) causes highest performance overhead (around 40%) compared to the lowest overhead case study which is Scenario A. As shown in Figure 4, by reducing the frequency scaling range (Scenarios B2 and B3), the performance overhead decreases to 23% and 18% highlighting the effectiveness of adapting frequency in lower range for efficient SCA detection and mitigation. Another interesting observation is that the DCU IP prefetcher in Scenario C2 has shown more influential performance reduction to applications' performance as compared to three other Scenarios (C1, C3, and C4). As depicted, under C2, performance overhead is 27% while the remaining three prefetcher adaptation scenarios achieve nearly 20% overhead. On the other hand, Scenario D has shown slightly higher overhead than adapting DCU IP prefetcher (C2) because applications are executed in high (3200MHz) and low frequency (2600MHz) with substantial frequency gap in between. Compared to closest state-of-the-art work [12], our proposed randomization methodology in this work based on frequency scaling and prefetchers adaptation, achieves significantly lower performance overhead, reducing the overhead from 32.66% to around 20% of performance loss. Furthermore, the proposed randomization-based solution can be effectively adopted only when victim applications are executed which are the target of side-channel attacks.

V. Conclusion

In this work first we analyze cache-based side-channel attacks relying on cache access time to identify the cache patterns of victim applications. We then thoroughly investigate system (frequency) and architecture (hardware prefetchers) impacts on cache access time to evaluate the potential parameters to adapt against the SCAs. Based on our comprehensive analysis of cache access pattern observations, we propose to randomly scale frequency and adapt hardware prefetchers to effectively mitigate the threats of SCAs. Furthermore, L3 Prime+Probe is deployed as a case study to evaluate the effectiveness of the proposed randomization-based SCA mitigation methodology. The experimental results indicate that under randomization of the frequency with the largest range (Scenario B1), building eviction set shows up to 50% failure rate which results in no sensitive operation sequence of victim application to be observed. Furthermore, scaling frequency with smaller ranges (Scenarios B2 and B3) or adapting prefetchers shows to be more effective on probing results with only 20% performance overhead. Our proposed randomization methodology outperforms the state-of-the-art SCA mitigation solutions achieving significantly lower performance overhead by reducing the overhead from 32.66% to closely 20% of performance loss.

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] F. Brasser and et al. Advances and throwbacks in hardware-assisted security: Special session. In *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, CASES '18, pages 15:1–15:10, Piscataway, NJ, USA, 2018. IEEE Press.

[3] Tien-Fu Chen and Jean-Loup Baer. Effective hardware-based data prefetching for high-performance processors. *IEEE transactions on computers*, 44(5):609–623, 1995.

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

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

[6] Matthew R Guthaus and et. al. Mibench: A free, commercially representative embedded benchmark suite. In *Proceedings of the fourth WWC*, 2001.

[7] Intel Intel. ia-32 architectures software developer's manual, volume 3b: System programming guide. *Part*, 1:64, 2007.

[8] Paul Kocher and et.al. Spectre attacks: Exploiting speculative execution. *arXiv:1801.01203*, 2018.

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

[10] Fangfei Liu and et.al. Last-level cache side-channel attacks are practical. In *SP*, pages 605–622. IEEE, 2015.

[11] Fangfei Liu and Ruby B Lee. Random fill cache architecture. In *47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014.

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

[13] Kyle J Nesbit and James E Smith. Data cache prefetching using a global history buffer. In *10th HPCA'04*.

[14] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: the case of aes. In *Cryptographers' Track at the RSA Conference*, pages 1–20. Springer, 2006.

[15] D Page. Partitioned cache architecture as a ėide-channel defence mechanism. 2005.

[16] H. Sayadi and et al. 2smart: A two-stage machine learning-based approach for run-time specialized hardware-assisted malware detection. In *DATE'1)*.

[17] H. Sayadi and et al. 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)*, pages 1–6. IEEE, 2018.

[18] Steven P Vanderwiel and David J Lilja. Data prefetch mechanisms. *ACM Computing Surveys (CSUR)*, 32(2):174–199, 2000.

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

[20] Zhenghong Wang and Ruby B Lee. New cache designs for thwarting software cache-based side channel attacks. In *ACM SIGARCH Computer Architecture News*, 2007.

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

[22] Tianwei Zhang, Yinqian Zhang, and Ruby B Lee. Cloudradar: A real-time side-channel attack detection system in clouds. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 118–140. Springer, 2016.