Seclusive Cache Hierarchy for Mitigating Cross-Core Cache and Coherence Directory Attacks

Vishal Gupta*, Vinod Ganesan†, Biswabandan Panda*
Department of Computer Science and Engineering
* Indian Institute of Technology Kanpur, † Indian Institute of Technology Madras, India
vishal@cse.iitk.ac.in, vinodg@cse.iitm.ac.in, biswap@cse.iitk.ac.in

***Abstract*—Cross-core cache attacks glean sensitive data by exploiting the fundamental interference at the shared resources like the last-level cache (LLC) and coherence directories. Complete non-interference will make cross-core cache attacks unsuccessful. To this end, we propose a seclusive cache hierarchy with zero storage overhead and a marginal increase in on-chip traffic, that provides non-interference by employing cache-privatization on demand. Upon a cross-core eviction by an attacker core at the LLC, the block is back-filled into the private cache of the victim core. Our back-fill strategy mitigates cross-core conflict based LLC and coherence directory-based attacks. We show the efficacy of the seclusive cache hierarchy by comparing it with existing cache hierarchies.**

I. Introduction

The advent of practical and robust cache attacks [1] has created an essential need for the research community to look for active defenses. The primary focus is on Last-Level Caches (LLCs), as most practical attacks target LLCs shared between different cores. These LLC attacks fall into two categories: (i) *Conflict-based attacks* [2], where the attacker tries to learn the victim's access pattern by carefully orchestrating their data-accesses and observing conflicts, and (ii) *Flush-based attacks* [3], where the attacker tries to attack memory locations shared with the victim.

Flush-based attacks are the easiest to mount since it relies on shared data and clflush [3] instruction. Although very practical, with broad applications in many prominent attacks such as Spectre [1], flush-based attacks have severe limitations as it depends on the clflush instruction. Many cloud platforms turn off clflush support, thus disabling this attack altogether [4]. On the other hand, conflict-based attacks are an important class of attacks that rely on the fundamental *interference property* of caches (i.e.) caches have a finite capacity and different addresses conflict for the same cache set. This property makes conflict-based attacks harder to mitigate, and we focus on defense for this class of attacks in our work.

In many commodity processors, multi-level cache hierarchies are often inclusive, *i.e.*, all lower levels of the cache hierarchy are a super-set of its higher levels. This inclusive property is maintained by the back-invalidation operation, which ensures that a data-eviction from a lower-level cache (shared LLC) mandates an eviction of that data from all higher-level private caches. Thus, an attacker can efficiently orchestrate conflicts in the shared LLC, invalidating the sensitive data from all private caches and use that to extract sensitive information from the victim. A natural solution is to use non-inclusive caches that do not let shared cache accesses affect private cache states. However, recent research suggests that the cache coherence directories that hold information of all these non-inclusive private caches are inclusive, making directories the new attack surface [4]. Besides, it is also possible to attack high-speed interconnects in non-inclusive caches using Invalidate+Transfer [5]. Thus, conflict-based cache attacks are here to stay, requiring a fundamental solution to mitigate them.

Existing efforts that try to seal these timing attacks in the LLC largely fall into three categories *viz.* cache partitioning [6], cache randomization [7] and secure policies for caches [8]–[10]. However, existing efforts are limited [11] and are often impractical to implement . In this work, we propose a fundamental change to cache hierarchies that seal conflict-based cache and coherence directory attacks with minimal overheads.

We propose *seclusive caches*, a secure cache hierarchy, that breaks the fundamental LLC interference between the attacker and the victim by privatizing the victim cache line upon an eviction. The critical approach in our proposed hierarchy consists of a *back-fill strategy*, where any cross-core eviction of a cache line detected at LLC is filled back to the private L2 cache of the victim core. This simple *back-fill* breaks the interference between the attacker and the victim, rendering conflict-based, coherence directory-based, and interconnect-based attacks invalid.

Prior solutions propose strict constraints on the cache hardware (e.g. partitioning) to break the existing side-channel, whereas we provide privatization on-demand. To the best of our knowledge, this is the first paper that proposes an effective solution towards securing multiple attack surfaces like LLC, coherence directory, and interconnect. We experimentally show that a *seclusive cache hierarchy* seals conflict based LLC side-channel attacks and evaluate our proposal using ChampSim [12], a micro-architectural simulator, to experimentally show the performance trade-offs over the existing multi-level cache hierarchies.

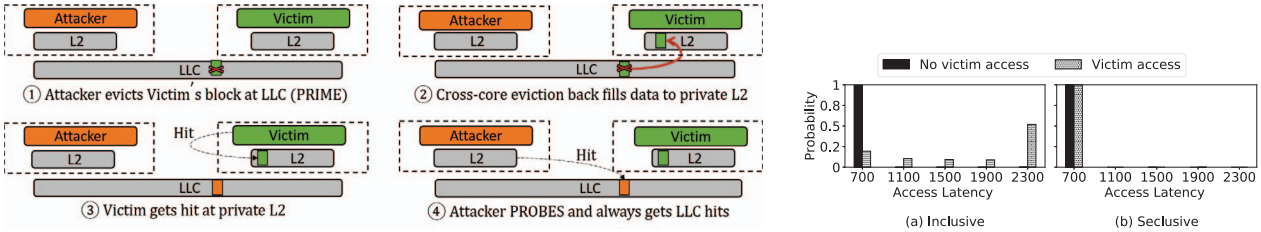
II. Background

A. Multi-level Cache Hierarchies

Multi-level cache hierarchies are designed to overcome the memory-wall problem. Interestingly, the multiple levels open up a novel design space based on the choice of data-flow

Full paper: https://www.cse.iitk.ac.in/users/vishal/seclusive_full_paper.pdf

978-3-9819263-5-4/DATE21/©2021 EDAA



(a) (b)

Fig. 1: (a) Eviction based attack on the Seclusive cache hierarchy. For simplicity, the L1 cache is not shown. (b) Probability distribution of attacker probe time (in cycles) in accessing the eviction set.

between the caches. Some popular design choices are described here.

Inclusive caches: For a cache-hierarchy to be inclusive, it should satisfy three objectives: (i) All lower-level caches should contain all the blocks of its higher-levels (super-set), (ii) A data-fetch from main-memory is filled to all the cache-levels upon a miss from all the cache-levels, and (iii) When a block is evicted from a cache-level, it should also be evicted from all its higher levels, termed *back-invalidation*.

Exclusive caches: This hierarchy is the opposite of inclusive caches where: (i) the lower cache-levels are not a super-set of its higher-levels, (ii) A data-fetch from main-memory is filled only to the cache in the highest-level upon a miss, and (iii) There is no *back-invalidation* upon an eviction of data from a cache-level.

Non-inclusive caches: This cache-hierarchy preserves the best of both inclusive and exclusive caches as follows: (i) Upon a cache-miss from all the levels, the fetched block is populated to all levels (inclusive), and (ii) there is no *back-invalidation* operation (exclusive).

B. Cache Timing Attacks

Typically, in a cross-core timing channel attack, the attacker exploits shared structures like LLC, cache coherence directory, and shared interconnect to leak information. These attacks can be broadly classified into two buckets:

Conflict based attacks. These attacks rely on caches having finite sizes and addresses will always contend for locations. A popular example is PRIME+PROBE [2], in which the attacker forms a set of cache-lines, called the *e*viction set to determine the victim's accesses which are often a function of data (e.g. secret-key) and can be revealing.

Flush based attacks. These attacks work by sharing some part of memory space with the victim process. The attacker flushes a shared block, and then waits for victim access. Then the attacker reloads the flushed block, and the access latency of that load reveals whether the victim has accessed that block or not. An example of such an attack is Flush+Reload [3].

C. Coherence Directory and Interconnect based Attack

A non-inclusive cache hierarchy makes cross-core LLC based attacks difficult as there is no back-invalidation to private caches. For invalidating the blocks from private caches, the attacker exploits the cache coherence directory [4] because of its inclusive property. For a directory structure such as the one present in Skylake-X server [4], there is a traditional directory that tracks the cache lines present in the LLC and an extended directory that tracks the cache lines present in the private caches.

This directory is inclusive in nature because it has to track all the blocks present in different levels of cache. The attacker creates contention in the extended directory (PRIME phase), which evicts the victim's block from its private L2 cache to the LLC. If the victim re-accesses that block, then it causes another contention in the extended directory, evicting one of the attacker's block to the LLC, which can be detected by the attacker during its PROBE phase, thereby leaking information. Thus, directories are the new inclusive attack surface in non-inclusive cache hierarchies.

Another cross-core attack [5] that works with non-inclusive cache hierarchy use high speed interconnect like AMD's HyperTransport, Intel QuickPath to leak secret information. The attack exploits the timing difference between accessing a block from a remote core's private cache and DRAM.

III. Seclusive Cache Hierarchy: Design Principles

The goal of a secure cache-hierarchy is to fundamentally seal side-channels using a lightweight solution (in terms of cost and area) with minimal performance degradation. In this section, we discuss our proposal called the seclusive cache hierarchy.

A. Working of Seclusive caches

Consider a three-level cache hierarchy with a multi-core system, having a shared LLC and private L1 and L2 caches. Let's assume there is a load request from a core for a cache block, which is not present in the entire cache hierarchy *i.e.* an LLC miss. The block is fetched from the DRAM and is filled into the LLC and requesting core's private caches. If the target LLC set is full, an existing block in the set has to be evicted based on the cache replacement policy. Two types of evictions are possible in this scenario: (i). The evicting and the evicted cache blocks are from the same core, and (ii). The evicting and the evicted cache blocks are from different cores (cross-core eviction). In *seclusive caches*, the same core evictions are handled similar to a non-inclusive LLC. However, on a cross-core eviction, the evicted block is *back-filled to the private cache (L2) of the core of evicted block*. This back-fill strategy privatizes the cache block and breaks the existing side channel. The cache coherence directory and additionally the presence of owner-bits in some implementation, can be used to detect cross-core evictions and the core to back-fill.

In case the evicted block is dirty, the dirty block is made clean by writing into the DRAM and subsequently the cleaned block is then back-filled to L2, with its dirty bit reset. In case the evicted block is shared between multiple cores, for instance with multi-threaded applications, the block gets back-filled to one of the sharers. If other threads require that block, then it can easily be serviced by a high speed interconnect. If it is not needed by any thread, then it can be safely evicted by that core, incurring minimum overhead.

B. *Effect of* Seclusive cache hierarchy *on Cross-Core Conflict based Cache Attacks*

Figure 1a shows the effect of cross-core conflict based attack with seclusive cache hierarchy. (①) The attacker evicts victim's block from the LLC. Since, this is a cross-core eviction (②), the block is back-filled to victim's cache. When the victim accesses the block (③), it gets a hit in its private L2, and does not accesses the LLC. When the attacker re-accesses its block, it gets hits all the time (④), giving an impression to the attacker that victim has never accessed its block. The attack is thus mitigated because the attacker does not get the true latency information of whether the victim has accessed its block or not since there is no observable interference at the LLC.

C. *Effect of* Seclusive cache hierarchy *on coherence directory and Interconnect based attacks*

In non-inclusive LLCs there is no back-invalidation operation, making conflict based cache attacks harder to mount. However, as discussed in section II-C the inclusive coherence directory becomes the new attack surface.

In a seclusive cache hierarchy, whenever the attacker tries to evict the victim's block using cross-core eviction in the extended directory, it back-fills the block to the victim's private cache, and we track that block in the traditional directory. When the victim re-accesses that block, it gets hit in its private L2, which breaks the reverse interference, which was the cause for the eviction of attacker's block. This breaks the side channel, as now attacker's accesses always result in a hit irrespective of the victim's accesses.

Interconnect based attacks work by evicting the victim's block from private caches and then measuring timing difference between remote core access and DRAM access, as mentioned in section II-C. In a seclusive cache hierarchy, when the attacker evicts victim's block using cross-core eviction, it back-fills the block to victim's private cache. Now, irrespective of the victim's accesses, the attacker always gets an access latency similar to the latency of remote core's access latency, breaking the side channel.

IV. Evaluation

Design Methodology: We evaluate seclusive cache hierarchy on a cycle-accurate, trace-based micro-architectural simulator, ChampSim [12], that models an out of order processor with multi-level caches and DRAM. The parameters for the simulated system are given in Table I. We model different L2-LLC cache sizes based on different Intel micro-architectures given in Table II. 32KB of L1I and 48KB of L1D is constant in all three systems.

TABLE I: Parameters of the simulated system.

Processor	8 cores, 4 GHz, out of order
L1D, L1I	48 KB (12 way), 32 KB (8 way), LRU
DRAM	4 DRAM controllers, 64 read/write queues, FR-FCFS, 3200MHz DRAM (11-11-11)

TABLE II: Cache Hierarchy Configuration

Micro-Architecture	L2 Size	LLC Size
Intel Ice-Lake	512 KB (8 way)	2 MB (16 way)
Intel Broadwell-EP	256 KB (8 way)	2.5 MB (20 way)
Intel Cascade Lake-SP	1024 KB (16 way)	1.375 MB (11 way)

A. *Security Analysis*

We analyse the security of different cache hierarchies with cross-core eviction based cache attacks. PRIME+PROBE attack, which has minimum assumptions, is chosen for our analysis. We run GnuPG 1.4.13 as our victim application which uses a private key to decrypt a message. In ChampSim, we pin the victim application to one core. Victim accesses the secret key dependent critical LLC sets containing square and multiply functions used in decrypting the message. The attacker is running on another core, implementing the attack on these critical LLC sets to recover the victim's private key.

Figure 1b show the probability distribution of attacker probe cycles when the victim accesses and does not access that critical cache set. For an inclusive cache hierarchy, when the victim does not access the critical LLC set, attacker observes a probe time of around 700 to 1100 cycles, whereas when victim accessed that critical LLC set, the probe time ranges from 700 to 2300 cycles, with 80% of probe time above 1100 cycles. This is because the attacker is able to evict victim's block due to back invalidation.

For the seclusive cache hierarchy, the attacker is not able to differentiate whether a victim has accessed a critical set or not. The probe time in both these cases is between 700 to 1100 cycles. This is because, upon a cross-core eviction from the attacker, the seclusive cache hierarchy back-fills the victim's block to the victim's private cache. When victim accesses that block, it gets hit in its private cache, and no effect on the LLC (victim does not evict the attacker's blocks). Hence, upon an attacker's PROBE on that LLC set, it gets all LLC hits. No threshold can be set to differentiate between the two cases effectively, thereby preventing the attack altogether. We observe similar latency distribution for cache coherence directory-based attacks with seclusive cache hierarchy.

B. *Performance Analysis*

The effect on performance with the use of seclusive cache hierarchy is compared with inclusive and non-inclusive cache hierarchies for 2, 4, and 8 core systems. Workloads used for simulation are SPEC CPU 2017 benchmarks and multi-threaded Client/Server traces from IPC-1. The traces are divided into LLC fitting and LLC thrashing applications, according to LLC Misses per Kilo Instructions (MPKI). Benchmarks having LLC MPKI greater than five are considered as thrashing applications and benchmark having LLC MPKI less than one are considered as fitting applications.

Evaluation is done on mixing these different benchmarks for different core counts. 14 mixes are generated with different

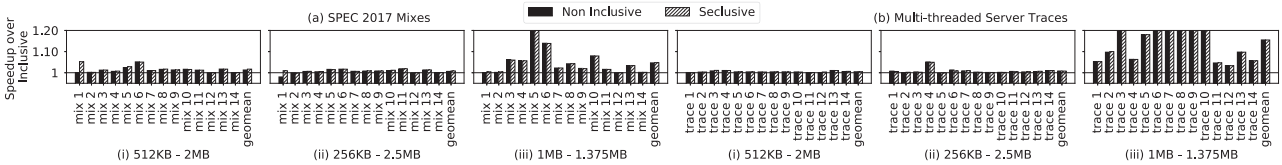


Fig. 2: Performance improvement in (a) SPEC 2017 mixes and (b) multi-threaded server traces in an 8-core system with different L2-LLC cache sizes.(higher the better).

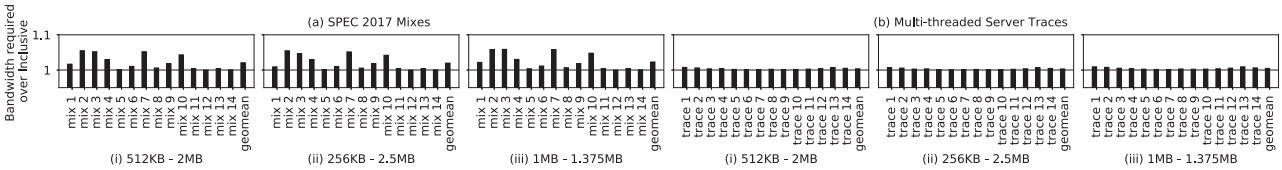


Fig. 3: Bandwidth requirement in (a) SPEC 2017 mixes and (b) multi-threaded server traces in an 8-core system with different L2-LLC cache sizes.(lower the better).

combinations of thrashing and fitting applications. Weighted speedup is used as a metric for evaluating performance in multi-core simulations.

Figure 2 show the performance (in terms of weighted speedup) of non-inclusive and seclusive normalized to inclusive cache hierarchy for SPEC and multi-threaded server traces. Figure 2 show that for seclusive cache hierarchy, there is no performance degradation as compared to the non-inclusive cache hierarchy. Due to space limitations, we are not showing the performance of seclusive directories, but it performs similarly to the seclusive cache hierarchy.

C. On-Chip Traffic Analysis

Modern processors have a banked LLC to improve bandwidth. Each bank is associated with a core, and all are connected to high speed interconnect. For an 8 core system, the LLC is 8 way banked and connected to an interconnect running at 4 GHz with a data bus of 256bits. The bandwidth between LLC and interconnect comes out to be 1 TB/s (256 bits per cycle$\times$cycles per second$\times$LLC bank count). Similarly, for 2 core and 4 core systems, the bandwidth is 256 GB/s and 512 GB/s, respectively.

Figure 3 show the additional bandwidth required by seclusive cache hierarchy normalized to an inclusive cache hierarchy for SPEC and multi-threaded server traces. For an 8 core system, the effective bandwidth required with seclusive cache hierarchy is less than 5% for most of the cases. In general, a thrashing application demands more bandwidth as compared to a fitting application.

D. Storage Overhead Analysis

Seclusive cache hierarchy has zero hardware storage overhead as it utilizes the coherence directory already present in modern multi-core systems. These directories store the information about which core is the owner of each cache block. During a cross-core eviction at the LLC, the back-fill mechanism uses this information to fill the victim cache block back to the owner core's private cache.

V. Conclusion

In this paper, we introduced the Seclusive cache hierarchy, a low overhead solution for mitigating cross-core last-level cache and coherence directory-based attacks. The performance is at par with non-inclusive cache hierarchy. The security guarantees are stronger than both inclusive and non-inclusive cache hierarchy as it prevents not only cross-core LLC attacks but also coherence directory, and interconnect-based attacks.

VI. Acknowledgements

We would like to thank all the anonymous reviewers for their helpful comments and suggestions. We would also like to thank members of CAR3S research group for their feedback on the initial draft. This work is supported by the SRC grant SRC-2853.001.

References

[1] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, *et al.*, "Spectre attacks: Exploiting speculative execution," in *IEEE S&P, 2019*.
[2] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *IEEE S&P '15*.
[3] Y. Yarom and K. Falkner, "FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack," in *USENIX Security '14*.
[4] M. Yan, R. Sprabery, B. Gopireddy, C. Fletcher, R. Campbell, and J. Torrellas, "Attack directories, not caches: Side channel attacks in a non-inclusive world," in *IEEE S&P'19*.
[5] G. Irazoqui, T. Eisenbarth, and B. Sunar, "Cross processor cache attacks," in *ASIA CCS '16*.
[6] V. Kiriansky, I. Lebedev, S. Amarasinghe, S. Devadas, and J. Emer, "DAWG: A defense against cache timing attacks in speculative execution processors," in *MICRO '18*.
[7] M. K. Qureshi, "CEASER: Mitigating conflict-based cache attacks via encrypted-address and remapping," in *MICRO '18*.
[8] M. Yan, B. Gopireddy, T. Shull, and J. Torrellas, "Secure hierarchy-aware cache replacement policy (SHARP): Defending against cache-based side channel attacks," in *ISCA '17*.
[9] M. Kayaalp, K. N. Khasawneh, H. A. Esfeden, J. Elwell, N. Abu-Ghazaleh, D. Ponomarev, *et al.*, "RIC: Relaxed inclusion caches for mitigating llc side-channel attacks," in *ACM/EDAC/IEEE DAC '17*.
[10] B. Panda, "Fooling the Sense of Cross-core Last-level Cache Eviction based Attacker by Prefetching Common Sense," in *PACT '19*.
[11] D. Kumar, C. S. Yashavant, B. Panda, and V. Gupta, "How Sharp is SHARP?," in *WOOT@USENIX Security '19*.
[12] Champsim. https://github.com/ChampSim/ChampSim.