ISSN: 2278-4632
Vol-10 Issue-3 No.01 March 2020

CROSS-CORE CACHE AND COHERENCE DIRECTORY ATTACKS ARE REDUCED BY USING A SECRET CACHE HIERARCHY

Mr.*Gyana Prakash Bhuyan*[1]*, *Mrs.Pragyan Paramita Panda*[2]
[1]**Assistant Professor,Dept. Of Computer Science and Engineering, NIT , BBSR*
[2]*Assistant Professor,Dept. Of Computer Science and Engineering, NIT , BBSR*
gyanprakash@thenalanda.com, pragyanparamita@thenalanda.com*

***Abstract*— By taking advantage of the inherent interference at shared resources like the last-level cache (LLC) and coherence directories, cross-core cache attacks are able to gather sensitive data. Cross-core cache attacks will fail in the absence of any interference. To achieve this, we suggest a secretive cache hierarchy that uses cache-privatization on demand to ensure non-interference while having no storage overhead and only a slight increase in on-chip traffic. The block is back-filled into the victim core's private cache following a cross-core eviction by an attacking core at the LLC. Cross-core conflict-based LLC and coherence directory-based assaults are lessened by our back-fill method. By contrasting it with current cache hierarchies, we demonstrate the seclusive cache hierarchy's effectiveness.**

I. INTRODUCTION

Multi-level cache hierarchies are indispensable for high- performance computer architectures. They alleviate long la- tency main-memory requests by storing frequently used data near the processor. However, from the perspective of security, caches have become an active attack surface to extract critical information, including private keys, keyboard strokes, and websites visited. This attack is made possible by observing the time difference between a hit and a miss, which is often relatedto an application's data access pattern.

The advent of practical and robust cache attacks [1], [2] 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 [3]. These LLC attacks fall into two categories:
(i) *Conflict-based attacks* [4], 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* [5], 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. The attacker uses clflush [5] instruction to evicta set of data blocks from the cache and observe the victim's accesses to those blocks. Although very practical, with broad applications in many prominent attacks such as Spectre [1], flush-based attacks have severe limitations as it depends onthe clflush instruction. Many cloud platforms turn off clflush support, thus disabling this attack altogether [6]. 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 allprivate 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 [6]. Besides, it is also possible to attack high-speed interconnects in non-inclusive caches using Invali- date+Transfer [7]. 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 [8]–[13], cache randomization [9], [14]–[16] and secure poli- cies for caches [17], [18]. However, existing efforts are limited and are often impractical to implement (high performance overhead, dependency on Instruction Set Architecture (ISA), runtime system, or Operating System (OS)). In this work, we propose a fundamental change to cache hierarchies thatseal 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. We make an important observation that practical conflict-based attacks are made possible due to cross-core evictions, where the attacker and the victim reside on different cores. 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 in- valid.

Prior solutions propose strict constraints on the cache hard-
ware (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 showthat a *seclusive cache hierarchy* seals conflict based LLC side- channel attacks and evaluate our proposal using Champsim [19], a micro-architectural simulator, to experimentally show the performance trade-offs over the existing multi-level cache hierarchies.

In summary, our key contributions are as follows:

- We propose *seclusive caches*, a secure multi-level cache hierarchy that prevents cross-core conflict-based LLC andcoherence directory attacks. (Section III)
- We show the efficacy of our proposed seclusive cache hierarchy by comparing its performance and security trade-offs with the currently existing multi-level cachehierarchies and other secure cache hierarchy proposals in the literature. (Section IV)

II. BACKGROUND

A. *Multi-level Cache Hierarchies*

Due to the ever-growing memory wall, multi-level cache hierarchies were implemented to reduce the latency gap. How- ever, these hierarchies open up a novel design space in which they can be designed in many different ways depending on where and how data flows between these caches. Some of the existing hierarchies that dictate the flow are as follows:

Inclusive cache hierarchy. As the name suggests, in inclu- sive cache hierarchy all the lower levels of the memory (L2, LLC, etc.) are a super-set of their higher levels. A miss ina higher level of the cache hierarchy permeates to its lower levels sequentially, until the block is found at any of the lower- levels of the cache hierarchy. In case it misses in all the levels, it is fetched from the main-memory and filled to all the cache levels to maintain inclusivity. In addition, this inclusive property mandates that whenever a block is evicted from any cache, the same block is evicted from all the higher levels of the cache-hierarchy, terned as *back-invalidation*. Inclusive caches are widely used because maintaining cache coherence is simple. However, the inclusive nature of the shared caches makes it a target for information leakage through side-channels.

Exclusive cache hierarchy. In this hierarchy, unlike theinclusive counterpart, the lower levels of caches are not a super- set of their higher levels *i.e.* each cache level has a unique set of data, and hence, exclusive cache hierarchy has higher effective cache capacity. Upon a miss from all the cache levels in an exclusive hierarchy, the block is fetched and populated only to the private caches present in the highest-level. The cache blockmarches down the hierarchy (L2 & LLC) as and when

ISSN: 2278-4632
Vol-10 Issue-3 No.01 March 2020

it is evicted from an upper-level cache.

Non-Inclusive cache hierarchy. It is a hybrid of both inclusive and exclusive cache hierarchies (neither inclusive nor exclusive). In this hierarchy, whenever there is a cache miss from all the cache levels, the fetched block is populated to all the levels (similar to inclusive cache hierarchies). However,whenever there is an eviction in the lower-level cache, the higher levels are not back-invalidated (similar to exclusive cache hierarchies). Hence, non-inclusive caches preserve the best of both inclusive and exclusive cache hierarchies.

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. One of the popular conflict based attacks is PRIME+PROBE [4]. It works by forming an eviction set, where the attacker finds W cache lines that fills a W-way cache set. Upon victim's access to that set, one of the attacker's W cache lines is replaced. By measuring the access time for reaccessing these W cache lines, the attacker gets information about the victim's accesses. These victim accesses are often a function of the data (e.g., accessing indexes based on a secret key in cryptographic algorithms), and hence these access informations can be used to steal critical data.

Flush based attacks. These attacks work by sharing some part of memory space with the victim process (e.g., shared library such as glibc). The attacker flushes a shared block (often using clflush), 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 [5].

Cross-core timing channel attacks at the LLC are more powerful than intra-core attacks at the private cache because LLC is shared between all the cores. Cross-core cache attacks are possible only at the LLC. For intra-core attacks, private caches are used because they are less susceptible to noise and give better accuracy. But, in a server environment, it is easier to get the attacker and victim process to share an LLC than a private cache. This paper focuses on preventing cross-core attacks because they require the least assumptions.

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 [6] because of its inclusive property. For a directory structure such as the one present in Skylake-X server [6], 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.

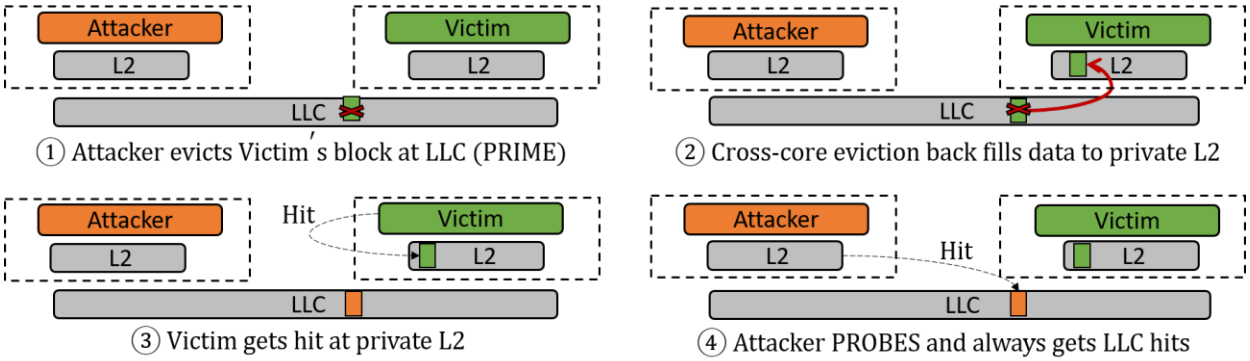


Fig. 1. Eviction based attack on the Seclusive cache hierarchy. For simplicity, the L1 cache is not shown.

Another cross-core attack [7] that works with non-inclusive cache hierarchy use high speed interconnect like AMD's Hyper-Transport [20], Intel QuickPath [3] to leak secret information. The attack exploits the timing difference between accessing a block from a remote core's private cache and DRAM. The attacker invalidates the blocks from the victim's cache. Whenever the victim accesses that block, it gets filled in the victim's private cache. When attacker accesses that block again, it gets less latency as compared to bringing that block from DRAM due to high speed interconnect.

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. The impending challenges in achieving this goal are threefold: (i) the design complexity of the solution, (ii) performance vs. security trade- off, and (iii) implications to cache coherence and directory- based attacks. In this section, we discuss our proposal called the seclusive cache hierarchy in detail and explain how it overcomes the aforementioned design challenges.

A. *Working of Seclusive caches*

Consider a three-level cache hierarchy with a multi-coresystem, 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 evictingand 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 itcan 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*

In an inclusive cache hierarchy, with a PRIME+PROBE attack, the attacker evicts victim's

ISSN: 2278-4632
Vol-10 Issue-3 No.01 March 2020

block at LLC by fillingthat cache set with its own data (PRIME phase). When the victim's block is evicted from the LLC, it is also evicted from its private caches due to back-invalidation. When the victim accesses that block again, it evicts an attacker's block at LLC, thereby leaking information to the attacker, during its PROBE phase.

Figure 1 shows the effect of cross-core conflict based attack with seclusive cache hierarchy. (1) The attacker evicts victim'sblock 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, anddoes 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.

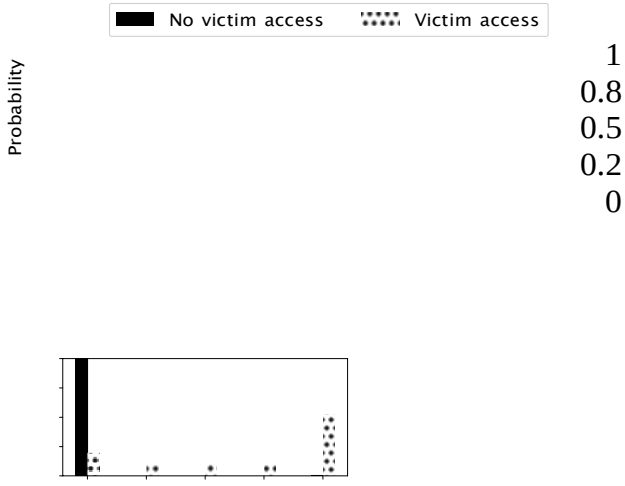
C. *Effect of* Seclusive cache hierarchy *on coherence directory and Interconnect based attacks*

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

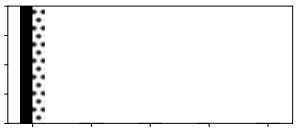
In a seclusive cache hierarchy, whenever the attacker tries to evict the victim's block using cross-core eviction in the extended directory, it back-fills the block to the victim's private cache, and we track that block in the traditional directory. Whenthe victim re-accesses that block, it gets hit in its private L2,

TABLE I
PARAMETERS OF THE SIMULATED SYSTEM.

Processor	2/4/8 cores, 4 GHz, out of order
L1D, L1I	48 KB (12 way), 32 KB (8 way), LRU
Cache line size	64 B in L1, L2 and L3
DRAM controller	1/2/4 controllers for 2/4/8-cores, 64 read/write queues, FR-FCFS
DRAM	3200 MHz (11-11-11)



(a) Inclusive



(b) Seclusive

TABLE II

ISSN: 2278-4632
Vol-10 Issue-3 No.01 March 2020

CACHE HIERARCHY CONFIGURATION

Fig. 2. Probability distribution of attacker probe time (in cycles) in accessing the eviction set.

Micro-Architecture	L2 Size	LLC Size
Intel Ice-Lake [24]	512 KB (8 way)	2 MB (16 way)
Intel Broadwell-EP [25]	256 KB (8 way)	2.5 MB (20 way)
Intel Cascade Lake-SP [25]	1024 KB (16 way)	1.375 MB (11 way)

TABLE III
REPRESENTATIVE WORKLOAD MIX TYPES

Mixes	Description
Mix 1	Same thrashing benchmark on all cores
Mix 2 - Mix 4	Different thrashing benchmark on all cores
Mix 5 - Mix 10	Mix of different fitting and thrashing benchmark
Mix 11	Same fitting benchmark on all cores
Mix 12 - Mix 14	Different fitting benchmark on all cores

ISSN: 2278-4632
Vol-10 Issue-3 No.01 March 2020

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 hierar- chy[1] on a cycle-accurate, trace-based micro-architectural simu- lator, ChampSim [19], that models an out of order processor with multi-level caches and DRAM. It has been recently used for evaluating LLC replacement policies at Cache Replacement Championship (CRC-2) [21], ISCA'17, data prefetchers at Third Data Prefetching Championship (DPC-3) [22], ISCA'19 and instruction prefetchers at First Instruction Prefetching Championship (IPC-1) [23], ISCA'20. 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.

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 [26] 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 PRIME+PROBE attack on these critical LLC sets to recover the victim's private key.

[1]For the benefit of the community, we will share the source code of seclusive cache hierarchy.

Figure 2 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. When victim reloads that block, the attacker gets an LLC miss for some of its blocks during its PROBE phase. Setting a threshold of 1100 cycles helps the attacker in differentiating between these accesses, thereby leaking the secret.

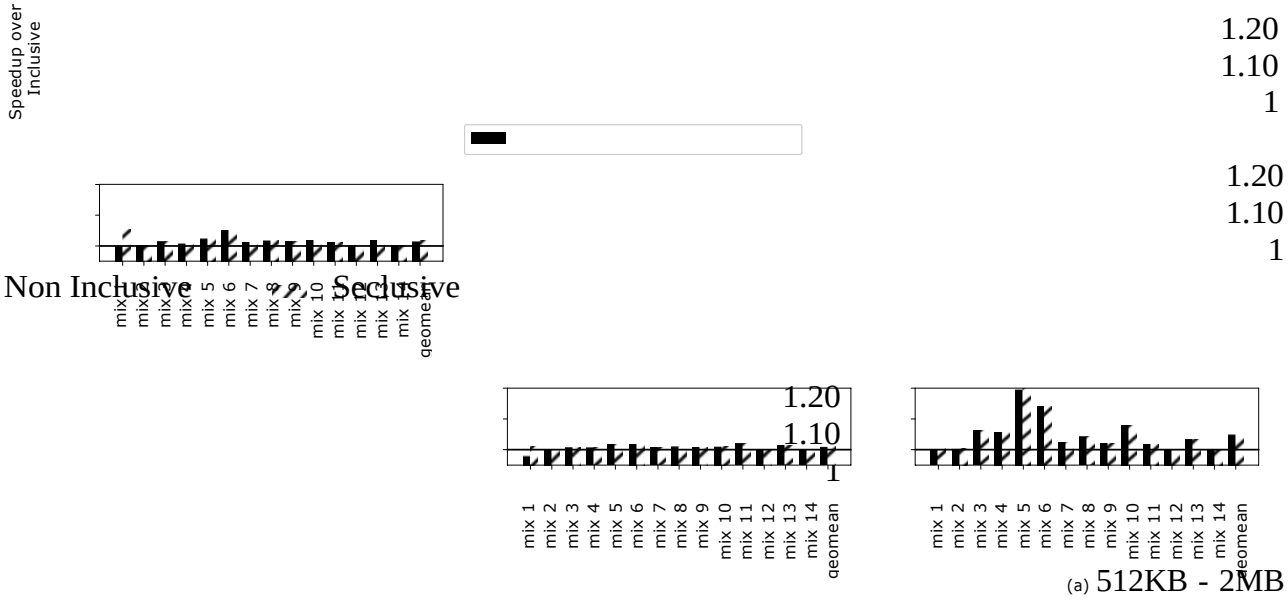
For the seclusive cache hierarchy, the attacker is not able to differentiate whether a victim has accessed a critical set or not. The probe time in both these cases is between 700 to 1100 cycles. This is because, upon a cross-core eviction from the attacker, the seclusive cache hierarchy back-fills the 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 [22], [27] and multi-threaded Client/Server traces from IPC-1 [23]. The traces are divided into LLC fitting and LLC thrashing applications, according to LLC

ISSN: 2278-4632
Vol-10 Issue-3 No.01 March 2020

Misses per Kilo Instructions (MPKI). Bench- marks having LLC MPKI greater than five are considered as thrashing applications and benchmark having LLC MPKI less than one are considered as fitting applications.

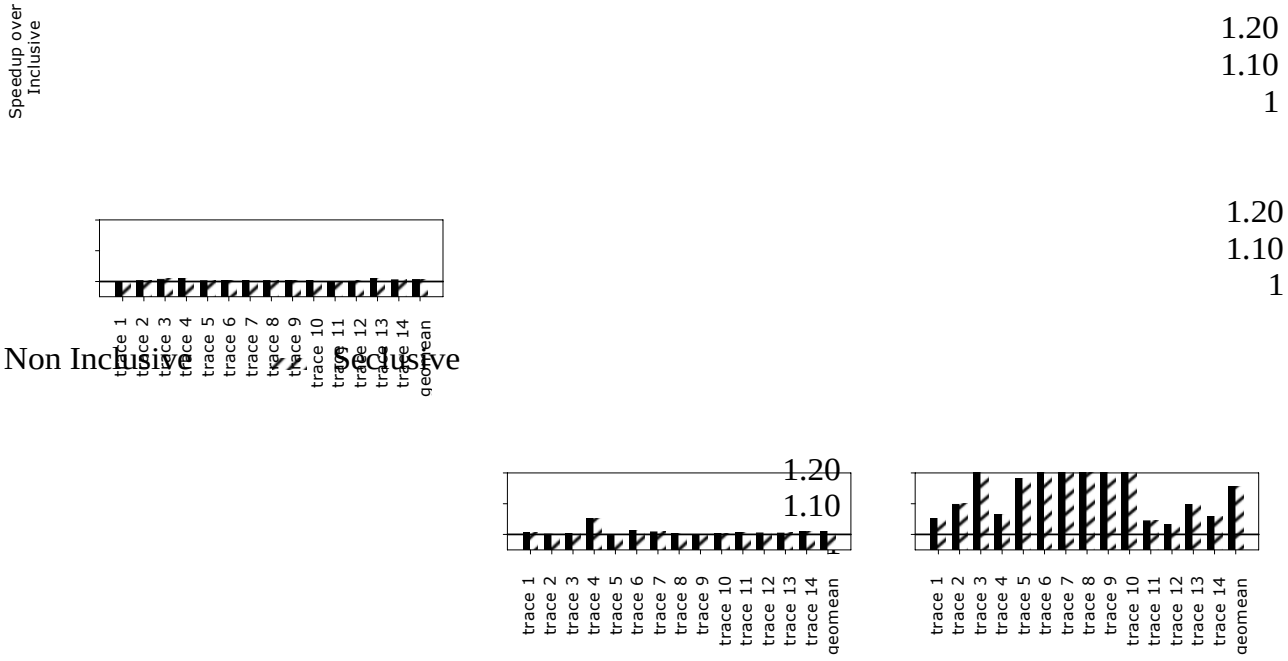


(a) 512KB - 2MB

(b) 256KB - 2.5MB

(c) 1MB - 1.375MB

Fig. 3. Performance improvement in SPEC 2017 mixes in an 8-core system with different L2-LLC cache sizes (higher the better).



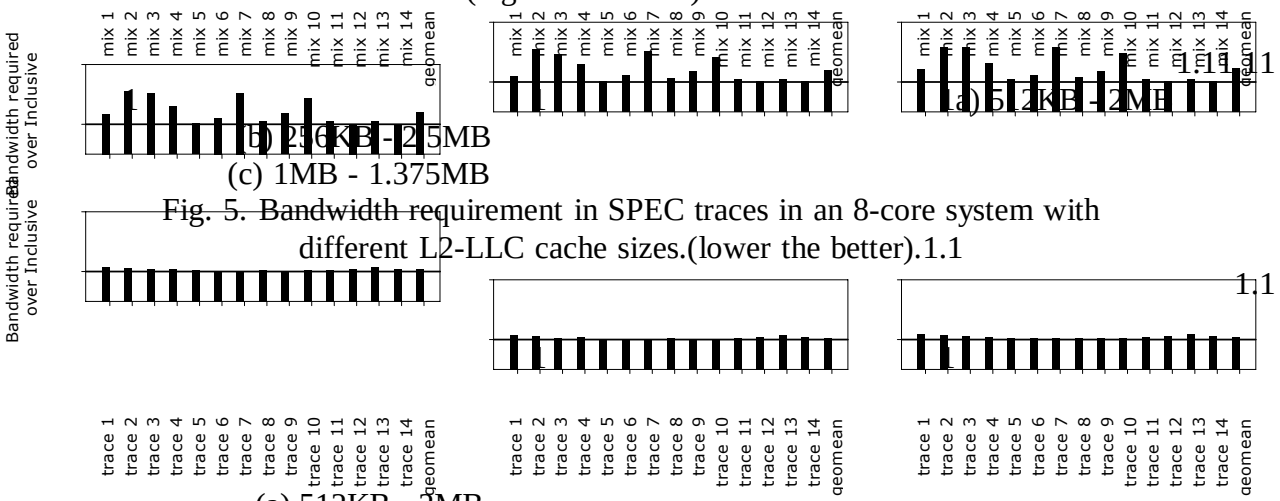
(a) 512KB - 2MB

(b) 256KB - 2.5MB

(c) 1MB - 1.375MB

ISSN: 2278-4632
Vol-10 Issue-3 No.01 March 2020

Fig. 4. Performance improvement in multi-threaded server traces in an 8-core system with different L2-LLC cache sizes.(higher the better).



(a) 512KB - 2MB
(b) 256KB - 2.5MB
(c) 1MB - 1.375MB

Fig. 5. Bandwidth requirement in SPEC traces in an 8-core system with different L2-LLC cache sizes.(lower the better).

(a) 512KB - 2MB
(b) 256KB - 2.5MB
(c) 1MB - 1.375MB

Fig. 6. Bandwidth requirement in multi-threaded server traces for an 8-core system with different L2-LLC cache sizes.(lower the better).

Evaluation is done on mixing these different benchmarks for different core counts. 14 mixes are generated with dif- ferent properties. Table III describes the representative mixes used for evaluating performance of different cache hierarchies. Weighted speedup [28] is used as a metric for evaluating performance in multi-core simulations. $WeightedSpeedup = \sum_{i=0}^{N-1} \frac{IPC_i^{together}}{IPC_i^{alone}}$, where $IPC_i^{together}$ is the IPC of core i when it is running with other N-1 application. IPC_i^{alone} is the IPC of core i when it is running alone in an N core system.

C. On-Chip Traffic Analysis

Modern processors have a banked LLC to improve band- width. Each bank is associated with a core, and all are con- nected 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 × cycles per second × LLC bank count). Similarly, for 2 core and 4 core systems, the

Figure 3 and Figure 4 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 3 and Figure 4 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.

ISSN: 2278-4632
Vol-10 Issue-3 No.01 March 2020

bandwidth is 256 GB/s and 512 GB/s, respectively.

Figure 5 and Figure 6 show the additional bandwidth re- quired 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 over- head as it utilizes the coherence directory already present in modern multi-core systems. These directories store the informa- tion 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.

E. Comparison with Relaxed Inclusion Cache

Relaxed Inclusion Cache (RIC) [18] was proposed as asecure cache hierarchy against LLC side-channel attacks, butit does not mitigate cross-core coherence directory and inter-connect attacks. It works by relaxing the inclusive property ofcache-hierarchy to prevent back-invalidation of private caches.It requires OS support to identify read-only and thread privatepages and requires a bit per cache block to track these relaxedblocks, which increase the overhead tremendously for largeLLCs. It also requires flushing of cache blocks when a threadis migrated or a page is swapped out, incurring a delay inthe critical path. Our proposal does not require any additionalstorage or support from OS, runtime system, ISA, or compiler.The high-performance numbers reported in RIC are because RIC has used a small LLC (2MB) for large core count (8cores), whereas the industry standard is 2MB/per core. Theperformance of RIC is below the non-inclusive cache hierarchy,and it decreases more when we increase LLC size with respectto the L2 cache size. This observation is also made in one of therecent works [29]. In comparison, the seclusive cache hierarchyperforms on the same scale and sometimes better than non-inclusive cache hierarchy, and does not degrade performance with different L2:LLC caches.

TABLE IV
CROSS-CORE ATTACKS WITH DIFFERENT CACHE HIERARCHIES.

Cache Hierarchy	Eviction based LLC Attacks	Coherence Directory and Inter-connect Attacks
Inclusive	Possible	Possible
Non-Inclusive	Not Possible	Possible
Relaxed Inclusion	Not Possible	Possible
Seclusive	Not Possible	Not Possible

Table IV summarizes whether cross-core eviction based LLCand coherence directory attacks are possible with different cache hierarchies.

V. RELATED WORK

Solutions like Cache Partitioning [8]–[13] separate the vic- tim's cache partition from the attacker's cache partition byeither partitioning the way, line, or set, or locking the linewith a protected bit. This enables non-interference between the two processes and breaks the side-

channel. However, the effective cache capacity is reduced, and it may even degrade the performance. In contrast, *seclusive cache hierarchy* break the side-channel without reducing the effective cache-capacity or impacting performance.

Randomization based solutions [9], [14]–[16] achieve non- interference by randomizing the address space by either having
permutation tables or fetching/evicting random lines or by encrypting the address and changing the key periodically. However, having permutation tables are not practical in a large LLC cache, and the encrypted caches protect from only conflict based attacks and not other attacks like coherence directoryor interconnect based attacks. In contrast, we propose a very practical solution that provides protection in different attack vectors like LLC, coherence directory, and interconnect.

Secure hierarchy aware replacement policy (SHARP) [17] is a proposal for mitigating cache-based attacks. It does so by changing the replacement policy to give priority to intra-core eviction and then if no choice is left, it goes for random cross- core eviction. The SHARP policy has a lot of loopholes and is not secure, as discussed in recent work [30].

VI. CONCLUSION

In this paper, we introduced the Seclusive cache hierarchy, a low overhead solution for mitigating cross-core last-levelcache 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.

VII. 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] Kocher *et al.*, "Spectre attacks: Exploiting speculative execution," *arXiv preprint arXiv:1801.01203*, 2018.
[2] Lipp *et al.*, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium (USENIX Security 18)*, pp. 973–990, 2018.
[3] An introduction to the QuickPath Interconnect. http://www.intel.com /content/dam/doc/white-paper/quick-path-interconnect-introduction- paper.pdf.
[4] Liu *et al.*, "Last-level cache side-channel attacks are practical," in *IEEE S&P '15*.
[5] Y. Yarom and K. Falkner, "Flush+reload: A high resolution, low noise, l3 cache side-channel attack," in *USENIX Security '14*.
[6] Yan *et al.*, "Attack directories, not caches: Side channel attacks in a non-inclusive world," IEEE S&P'19.
[7] Irazoqui *et al.*, "Cross processor cache attacks," in *ASIA CCS '16*.
[8] Osvik *et al.*, "Cache attacks and countermeasures: The case of aes," in *CT-RSA '06*.
[9] Z. Wang and R. B. Lee, "New cache designs for thwarting softwarecache-based side channel attacks," *ACM SIGARCH CAN '07*.
[10] Domnitser *et al.*, "Non-monopolizable caches: Low-complexity mitiga-tion of cache side channel attacks," *ACM TACO '12*.
[11] Kim *et al.*, "STEALTHMEM: System-level protection against cache-based side channel attacks in the cloud," in *USENIX Security '12*.
[12] Zhou *et al.*, "A software approach to defeating side channels in last-levelcaches," in *ACM SIGSAC CCS'16*.
[13] Kiriansky *et al.*, "DAWG: A defense against cache timing attacks inspeculative execution processors," in *MICRO '18*.

ISSN: 2278-4632
Vol-10 Issue-3 No.01 March 2020

[14] Liu *et al.*, "Newcache: Secure cache architecture thwarting cache side-channel attacks," *MICRO '16*.
[15] M. K. Qureshi, "Ceaser: Mitigating conflict-based cache attacks via encrypted-address and remapping," in *MICRO '18*.
[16] M. K. Qureshi, "New attacks and defense for encrypted-address cache," in *ISCA '19*.
[17] Yan *et al.*, "Secure hierarchy-aware cache replacement policy SHARP: Defending against cache-based side channel attacks," in *ISCA '17*.
[18] Kayaalp *et al.*, "RIC: relaxed inclusion caches for mitigating LLC side-channel attacks," in *ACM/EDAC/IEEE DAC '17*.
[19] Champsim. https://github.com/ChampSim/ChampSim.
[20] HyperTransport Technology white paper. http://www.hypertransport.org/docs/wp/ht system design.pdf.
[21] Second Cache Replacement Championship. https://crc2.ece.tamu.edu/.
[22] Third Data Prefetching Championship. https://dpc3.compas.cs.stonybrook.edu/.
[23] First Instruction Prefetching Championship. https://research.ece.ncsu.edu/ipc/.
[24] Intel Ice Lake microarchitecture, https://en.wikipedia.org/wiki/Ice Lake (microprocessor).
[25] C. L. Alappat *et al.*
[26] GnuPG 1.4.13, https://gnupg.org/ftp/gcrypt/gnupg/.
[27] Bucek *et al.*, "SPEC CPU2017: Next-generation compute benchmark," in *ACM/SPEC ICPE '18*.
[28] A. Snavely and D. M. Tullsen, "Symbiotic jobscheduling for a simulta-neous multithreaded processor," in *ASPLOS '09*.
[29] B. Panda, "Fooling the sense of cross-core last-level cache eviction based attacker by prefetching common sense," in *PACT '19*.
[30] Kumar *et al.*, "How sharp is SHARP ?," in *WOOT@USENIX Security '19*.