A Brief Survey of Cache Side Channel Attacks and Their Countermeasures

Shah Hassan, Maryam Babaie
Department of Computer Science
University of Central Florida
shahhassan@knights.ucf.edu, babaie@knights.ucf.edu

ABSTRACT

Cache side channel attacks exploit the inclusive property of shared last-level caches, and leak critical information through cache access timings of victim's program. Although these attacks have become more prevalent and adaptive over time; however, the prior art lacks a systematic analysis of their evolution and associated attack vector affinities. Motivated by this gap, in this paper, we briefly survey the recent cache side channel attacks and their countermeasures. For each attack vector, we explore its: 1) design constructs, 2) impact on the target system, and 3) possible deterrence schemes.

Keywords– *Caches, Side-Channel Attacks, Countermeasures, TLB, Page Table Walk, Translation Caches*

1. INTRODUCTION

With cloud computing, multiple virtual machines (VMs) share the underlying hardware resources, such as the processor (and its cores), memory hierarchy, memory, storage, and networks. While such sharing allows high resource utilization and reduces costs, it also creates *side channel* vulnerabilities, where sensitive information of a program leaks out through its usage behavior of hardware resources that are observable by the attacker. The attacker is assumed to run a program in a VM that shares the hardware resources as the victim's VM.

One type of shared hardware is the Last Level Cache (LLC). A typical processor has multiple cores, where each core has its own private level 1 and level 2 caches, but shares a level 3 cache with other cores. The LLC (or level 3 cache) is closest to the processor cores and has fine-grain sharing between cores, hence it is most vulnerable to side channel leakage that allows low-noise and high bandwidth side channel attacks.

The goal of this paper is to give a survey on side channel attacks on the shared LLC, including Prime+Probe, Evict+ Reload, Flush+Reload, and Prime+Abort, to name a few. Researchers have proposed mitigation using software or hardware techniques. Software-based page coloring technique allows the Operating System (OS) to partition the LLC horizontally by assigning specific cache sets to specific VMs. Intel Cache Allocation Technology (CAT) supports vertical partitioning of the LLC such that different VMs are allocated different ways of the cache. Recently, a new indirect cache attack was proposed which would bypass all existing mitigation techniques. The Xlate attack abuses hardware component in the Memory Management Unit (MMU) by interfering with Translation Caches (instead of the LLC) and measuring timing leakage of the victim. We aim to analyze all information related to the cache side channel attacks and their defenses.

The remainder of the paper is organized as follows. Section 2 provides background information on caches, page table, transactional memory, and Intel CAT. Section 3 provides details about cache side channel attacks. Section 4 presents countermeasures. Finally, Section 5 concludes the paper and Section 6 proposes some unanswered questions that exist in this research field.

2. BACKGROUND

2.1 CPU Caches

Modern processors use a hierarchy of caches, with level 1 (L1) cache being the smallest, fastest, and closest to the processor, while level 2 (L2) or level 3 (L3) cache being larger, slower, and farther away from the processor. Each of these caches is organized into sets (or rows), each containing multiple ways (or columns) of cache blocks/lines. A memory instruction, such as load/store, accesses the cache in three main steps. Firstly, index bits are used to locate the right set of the cache. Then, tags from all ways are read out for comparison with the tag bits of the accessed address. If they match (indicating a cache hit), the block offset bits are used to locate the right word/byte from the cache line. Modern Intel and AMD processors have 3 levels of caches; while several ARM processors have 2 levels of caches. Almost in all of these systems, the LLC is shared across cores, while upper level caches are private to each core. Figure 1 shows a 3-levels cache hierarchy and compares its characteristic in each level.

2.2 Page Table and Translation Lookaside Buffer

The addresses generated by a program in ld/st instructions are virtual addresses (VA) but main memory is managed by the OS using physical addresses (PA). Thus, VA needs to be translated into PA. The OS assigns and manages a page

Table 1: Comparison of popular cache side channel attacks.

Attacks	Memory Deduplication	Target	Same-core	Cross-core	Measurement
PRIME + PROBE	Not Required	Sets	Yes	Yes	Time
FLUSH + RELOAD	Required	Blocks	Yes	Yes	Time
FLUSH + FLUSH	Required	Blocks	Yes	Yes	Time
EVICT + RELOAD	Required	Blocks	Yes	Yes	Time
PRIME + ABORT	Required/Not Required	Sets/Blocks	Yes/No	Yes/No	TSX
Xlate	Not Required	Sets	Yes	Yes	Time/TSX

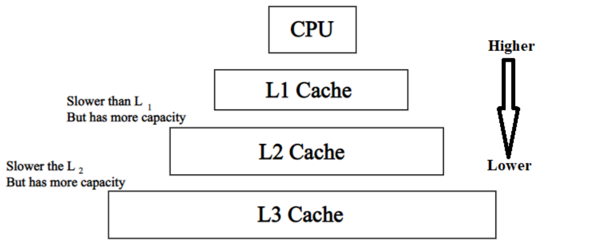
CPU
L1 Cache
L2 Cache
L3 Cache
Slower than L 1 But has more capacity
Slower the L 2 But has more capacity
Higher
Lower

Figure 1: Cache hierarchy levels in a modern processor, from top to bottom size of increases and speed decreases.

table for each process to perform the mapping of VA to PA. Since page table may be large and slow to access, modern processors use a cache to keep frequently needed entries, and such a cache is referred to as Translation Lookaside Buffer (TLB).

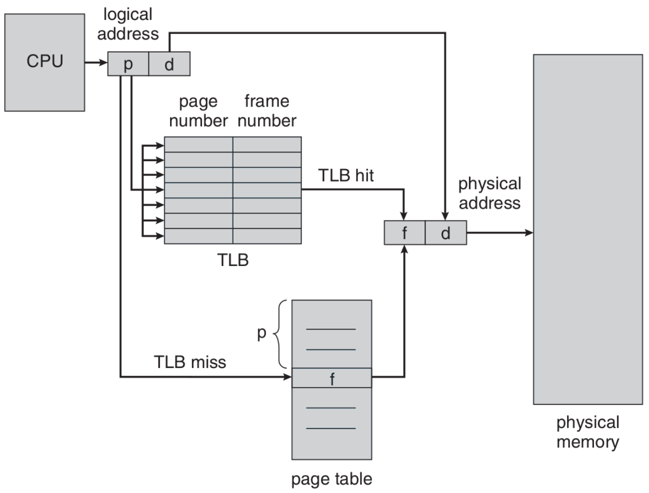

Figure 2: The addressing system in a modern processor and semantic functionality of Page Table and TLB [12].

2.3 Large Pages

A TLB miss hurts performance due to the need to traverse (or walk) the page table. A typical page table in x86 systems is hierarchical and to serve a single TLB miss, multiple memory lookups (one per hierarchy level) are needed. A hierarchy level of four or larger is typical for a 64-bit machine. To avoid large performance penalties associated with TLB misses, we can reduce the number of misses or the TLB miss penalty. To reduce the misses, the OS encourages the use of large pages. Normal pages have a size of 4KB; large pages may be upto 2MB in size or even larger [11]. Figure 3 provides a comparision of address decomposition between a small and large page.

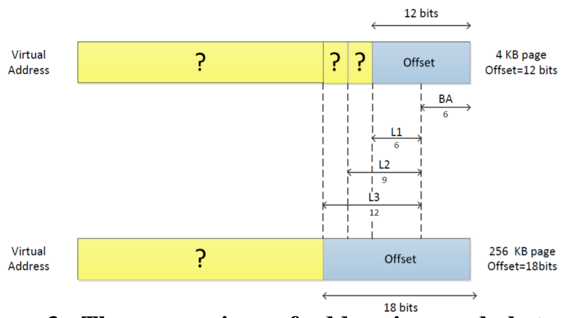

Figure 3: The comparison of addressing mode between when the page size is 4KB (top) and the page size is 256KB (bottom). As the page size grows, the number of bits needed to provide offset within that page grows as well; however, each page covers a larger portion of the memory space and the TLB hit rate will improve [13].

2.4 Page Table Walk and Translation Caches

To reduce the TLB miss penalty, ideally page table entries (PTEs) are cacheable in the L1/L2/LLC. However, they are still slow to access and since PTEs share the space with data, they may get evicted by data, causing slow TLB miss handling. To improve performance, specialized caches to hold PTEs were invented, including Translation Cache. A Translation Cache not only keeps PTEs (to avoid accessing main memory), it also caches the translation of VA to PA (condensing multiple PTE lookups into one). Figure 4 shows the precise decomposition of a virtual address by x86-64 processors [200]. Standard x86-64 pages are 4KB, so there is a single 12-bit page offset. The remainder of the 48-bit virtual address is divided into four 9-bit indices, which are used to select entries from the four levels of the page table. The four levels of the x86-64 page table are named PML4 (Page Map Level 4), PDP (Page Directory Pointer, L3), PD (Page Directory, L2) and PT (Page Table, L1). Finally, the 48-bit virtual address is sign extended to 64 bits [14].

63:48	47:39	38:30	29:21	20:12	11:0
se	L4 idx	L3 idx	L2 idx	L1 idx	*page offset*

Figure 4: Sub-fields of the x86-64 virtual address [14].

Figure 5 shows an example page walk for virtual address (0b9, 00c, 0ae, 0c2, 016). Each page table entry stores the physical page number for either the next lower level page table page (for L4, L3, and L2) or the data page (for L1). Only 12 bits of the 40-bit physical page number are shown in these figures for simplicity [14]. Moreover, figure 6 explains how these walks can be cached in the translation caches.

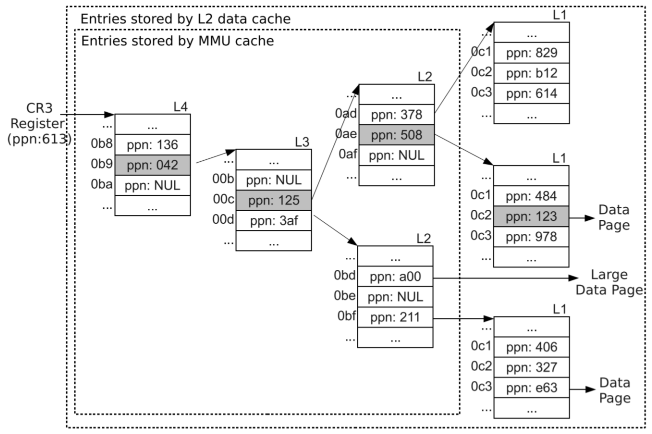

Figure 5: An example page walk for virtual address (0b9, 00c, 0ae, 0c2, 016) [14].

Base Location	Index	Next Page
125	0ae	508
042	00c	125
613	0b9	042
...	...	...

	Base Location	Index	Next Page
L2 entries	125	0ae	508
	...	...	...
L3 entries	042	00c	125
	...	...	...
L4 entries	613	0b9	042
	...	...	...

Figure 6: Two implementations of Translation Caches for virtual address (0b9, 00c, 0ae, 0c2, 016). In the above diagram, a unified Cache is implemented for all page table levels. In the below diagram, a different cache is implemented for different page table level [14].

2.5 Transactional Memory and TSX

Transactional Memory (TM) allows threads to execute in parallel even though they are not provably parallel. A transaction may abort if a conflict between threads is discovered. Intel TSX is Intel implementation of TM. TSX consists of instruction set extension that lets programmers to mark the beginning and end of a transaction.

2.6 Intel Cache Allocation Technology (CAT)

Intel introduced Cache Allocation Technology (CAT) in 2014. CAT is a hardware mechanism to partition LLC cache ways. CAT supports a number of classes of service (COS) and a bitmap which defines the LLC ways which a COS can access.

3. CACHE SIDE CHANNEL ATTACKS

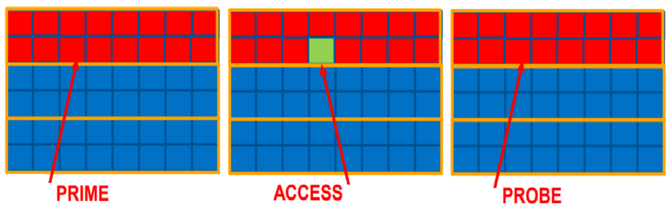

Figure 7: Prime+Probe attack on a cache. In the top, the attacker primes a buffer which is 2 sets size. In the middle, the victim accesses a block (in green). In the bottom, the attacker probes its buffer and the block accessed by the victim will be a miss for the attacker and attacker will find a information leak channel.[16]

3.1 Prime+Probe

Prime+Probe [1, 2] is an attack technique that allows attackers to learn about victim's secret by inferring the timing of cache set access. The attack has three main steps. First, the attacker primes one or more cache sets by executing its own program. Then, attacker waits for a pre-configured time while victim accesses the cache. As the victim brings in its data block, it evicts data blocks that the attacker primed. Finally, the attacker probes by measuring the time to load his data on each set, to check if victim's program has accessed any of the primed cache sets or not.

Before the introduction of shared LLC, Prime+Probe primarily targeted L1 Cache to monitor cache access from the same core running multiple threads. The inclusive nature of LLC allowed researchers to propose another Prime+Probe attack on LLC which enable the attack on cross-core and cross-VM setups.

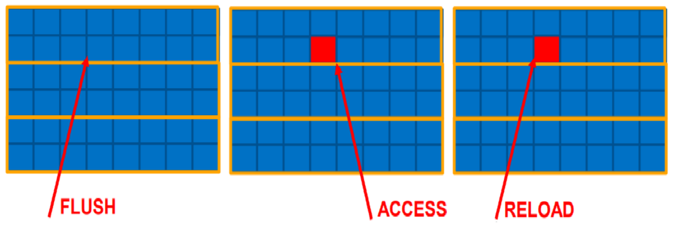

Figure 8: Flush Reload attack on Cache. In the top, the attacker flushes a cache block using clflush instruction. In the middle, the victim accesses that flushed block (in red). In the bottom, the attacker reloads the same block and the attacker will observe slow access as this would result in eviction of victim's block.[16]

3.2 Flush+Reload and Flush+Flush

The existence of memory deduplication may also lead to a side channel attacks on shared pages. The key idea of memory deduplication is, if multiple memory pages have the same content, then the memory manager only needs to keep one copy for these so-called deduplicated pages. Flush+Reload [3] attack exploits memory deduplication using *clflush* instruction; an intel instruction which can evict a specific memory block from all caches on chip. The attack is less noisier than Prime+Probe because it targets specific cache blocks. In Flush+Reload, attacker flushes one or more block(s) and waits for victim to access it. Attackers then reload these blocks and measure the access time to figure out whether victim has accessed them or not. A shorter access time indicates a cache hit, which indicates an earlier access by the victim. Like Prime+Probe, Flush+Reload attack can work with both cross-core and cross-VM systems.

Flush+Flush [4] is a variant of Flush+Reload attack and it relies only on a clflush instruction. Instead of reloading these blocks to check whether victim has accessed them, attacker now again flushes the block and measure time for the flush to complete. It is based on the observation that a flush completes faster if the block is not cached anywhere.

3.3 Evict+Reload

In the case of large pages, more number of bits are used as Page offset and hence attacker can exploit this and targets cache blocks on any cache levels. The Evict+Reload attack [1] relies on the shared memory and uses large pages to evict some blocks from last level cache. If Victim accesses any of these blocks, the attacker can figure this out and leak sensitive information about victim's input for the program. Like Flush+Reload, Evict+Reload attack also is less noisier than Prime + Probe and exploits the inclusive nature of shared LLC.

3.4 Prime+Abort

Prime+Abort [1] is a variant of Prime+Probe attack which relies on Intel′s Transactional Synchronization Extension (TSX). These attacks have no need of measuring time; instead it receives a Hardware Callback when victim program accesses cache sets primed by attacker. Although, the attack does not have to conduct some timing operation to determine victim's behavior but it has some structural weaknesses. The attack cannot monitor multiple targets; however multiple blocks can be targeted in the case of Flush+Reload and multiple sets can be targeted in the case of Prime+Probe. There are three variants of Prime + Abort attacks namely Naive TSX based Attack, Prime + Abort-L1 and Prime + Abort-L3. Briefly we can mention causes of transactional aborts in Intel TSX as follows:

- 1. Executing certain instructions, such as CPUID or the explicit XABORT instruction
- 2. Executing system calls
- 3. OS interrupts
- 4. Nesting transactions too deeply
- 5. Access violations and page faults
- 6. Read-Write or Write-Write memory conflicts with other threads or processes (including other cores) at the cache line granularity (whether those other processes are using TSX or not)
- 7. A cache line which has been written during the transaction (i.e., a cache line in the transaction write set) is evicted from the L1 cache
- 8. A cache line which has been read during the transaction (i.e., a cache line in the transactions read set) is evicted from the L3 cache

We will see that Causes 6-8 are used by attackers to get abort calls for different Prime + Abort Cache Side Channel attacks.

3.4.1 Naive TSX-based Attack

Naive TSX-based attack exploits one of the causes (Cause 6) of transactional aborts which is Read-Write (R-W) or Write-Write (W-W) Memory conflict with other processes or threads. Transactional aborts from R-W and W-W conflicts could lead to an attack implementation where attacker open a transaction, access target address and wait for an abort. The attack target precise Cache line/block and therefore require memory deduplication.

3.4.2 PRIME+ABORT-L1

Prime+Abort-L1 targets cache sets and on a Cache line eviction from the target set of L1 Cache, attacker can identify secret from the victim. Since L1 Cache is private to each core, the attacker can only spy on threads which share cores. Like Prime+Probe attack, this attack doesnt require memory deduplication. Attacker uses Cause 7 to get Abort call.

3.4.3 PRIME +ABORT-L3

Unlike Prime+Abort-L1 attack which targets cache sets from L1 Cache, Prime+Abort-L3 targets cache sets from shared L3 cache and therefore attacker can build eviction set from different core. Attacker can spy on both threads and processes level. Prime+Abort L3 doesnâĂŹt require memory deduplication. Attacker uses Cause 8 mentioned above to get abort call.

3.5 Xlate Attacks

As address from the software has to go through the Memory Management Unit (MMU) for VA-to-PA translation. As discussed earlier, the MMU consists of TLB and Translation Caches. Since the Translation Cache is shared by CPU and MMU, it is possible for the attacker to build an eviction set indirectly and monitor activities of victim's input of the program in a similar fashion as Prime+Probe. Researchers recently introduced new indirect cache attacks called Xlate [6] which can exploit the existence of Translation Caches and can leak sensitive information about victim′s input for the program. There can be three main kinds of Xlate attacks. 1) Xlate + Time, 2) Xlate + Probe and 3) Xlate + Abort.

3.5.1 Xlate+Time

In Xlate+Time attack, the attackers fill cache set with page table entries from eviction set and measure victims execution time to determine if victim accesses same cache set.

Table 2: Comparison of Prime + Abort Side Channel Attacks.

Attacks	Memory Deduplication	Target	Same-core	Cross-core
Naive TSX based Attack	Required	Blocks	Yes	Yes
Prime + Abort L1	Not Required	Sets	Yes	No
Prime + Abort L3	Not Required	Sets	No	Yes

Table 3: Comparison of Xlate Side Channel Attacks.

Attacks	Memory Deduplication	Target	Same-core	Cross-core	Measurement
Xlate + Time	Not Required	Sets	Yes	Yes	Time
Xlate + Probe	Not Required	Sets	Yes	Yes	Time
Xlate + Abort	Not Required	Sets	Yes	Yes	TSX

3.5.2 *Xlate+Probe*

In Xlate+ Probe attack, attackers can avoid measuring time of victims program by measuring the time taken to access attackers program repeatedly using the page table entries of eviction set.

3.5.3 *Xlate+Abort*

Xlate+Abort leverages Intels TSX by filling the cache set with the page table entries of eviction set within the transaction. If victim accesses the memory address that maps to same cache set, the transaction is likely to get an abort call.

4. COUNTERMEASURES

Because side-channel attacks rely on the relationship between information emitted (leaked) through a side channel and the secret data, countermeasures fall into two main categories: (1) eliminate or reduce the release of such information and (2) eliminate the relationship between the leaked information and the secret data, that is, make the leaked information unrelated, or rather uncorrelated, to the secret data, typically through some form of randomization of the ciphertext that transforms the data in a way that can be undone after the cryptographic operation (e.g., decryption) is completed [15]. The attacks described in the previous section depend on victim and attacker processes sharing the same cache resources. Mitigation can be designed to avoid such sharing, using software techniques such as Page Coloring and Cache Bar, or hardware techniques such as CAT and Autolock. We will describe them more in the next sections.

Table 4: Comparision of the Countermeasures

Countermeasure	Implementation	Method	System
Page Coloring	Software	Sets	Same & Cross-core
Cache Bar	Software	Sets/Lines	Cross-core
CATalyst	Hardware	Ways	Cross-core
ARM Autolock	Hardware	Pinning	Cross-core

4.1 Page Coloring

Page Coloring is software-based approach to make sure that two cross-core user processes land on different sets in the shared Last Level Cache [7]. Since PA is used to determine sets in the LLC, first few bits can be used as color to color sets of the LLC. This is shown in the top part of figure

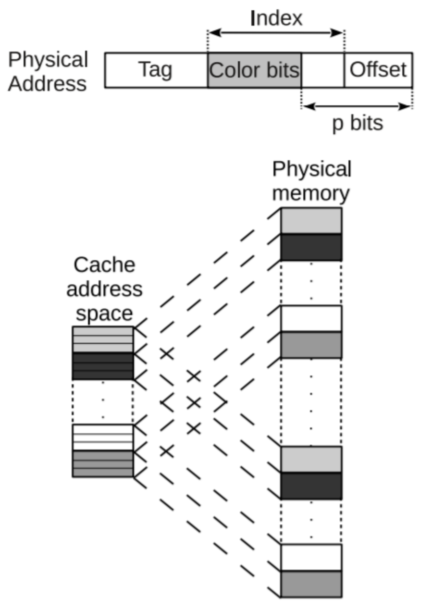

Figure 9: Page Coloring: on top the sub-fileds of the physical address, in order to support coloring, is shown. On the bottom, it is shown how the mapping between actual physical address space and cache address space would be done while supporting page coloring [7].

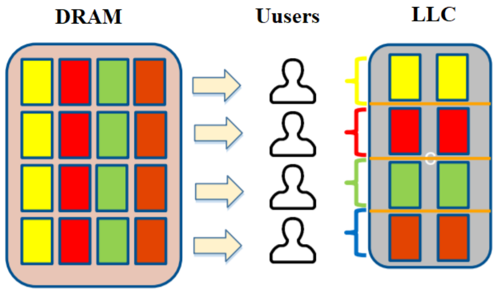

Figure 10: Page Coloring Scheme, here as there are 4 user cores, 2 bits have been used for coloring, so there are 4 colors. LLC is divided based on colors and each user can only access to the corresponding color in LLC [16].

9. In this way, specific sets can be assigned to each core resulting in the horizontal partitioning of LLC between cores and attacks can be mitigated, as it is shown in figure 10. An advantage of using this approach is that it does not require hardware to be modified and does not incur any hardware overheads.

4.2 Cache Bar

Cache bar [10] is a software-based mechanism to deter Flush+ Reload and Prime+ Probe side channel attacks. It provides a copy-on-access mechanism to manage shared physical memory pages and prevents Flush + Reload attack. It also provides another mechanism to dynamically maintain queues of cache-able memory pages to limit the cache blocks/lines a spy program may access in Prime + Probe attacks.

4.3 CATalyst

Intel Cache Allocation Technology can be used to vertically partition the LLC [8]. CAT was originally designed to eliminate the performance variation that results from cache sharing. It provides a hardware framework to allocate some cache ways to cores, providing separate partitioning of hardware resources between victim and attacker. Additionally, the OS may mark some cache ways as unevictable. If victim makes use of CAT and mark victim accesses as unevictable, attacker won′t be able to influence victim′s Cache accesses. Figure 11 shows a straight forward method of way-paritioning. As the figure shows one set has been divided into 4 parts (as there are 4 cores) and each of them is associated with one core.

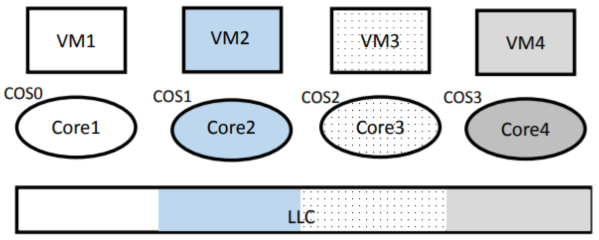

Figure 11: Way-partitioning (4 partitions) of the LLC between cores. [8]

4.4 ARM Autolock

Autolock [9] is a performance feature of inclusive caches in ARM, which allows recently used blocks to get locked from LLC if these blocks still exist on the higher level of cache. Using an inclusion bit, blocks can be marked as unevictable. With this in place, Attacker and Victim Processes would not be able to evict each other cache blocks from different cores.

5. CONCLUSION

In this paper, we explore various cache side channel attacks on shared LLC. Notably we made the following key observations: 1) PRIME + PROBE, EVICT + RELOAD, FLUSH + RELOAD, and FLUSH + FLUSH rely on measuring time to access chache lines; 2) PRIME + ABORT relies on Intel′s TSX to get abort call; and 3) Xlate Attack leaks victim′s information indirectly by exploiting Translation Caches. For each attack vector, we supplement our paper with existing countermeasures. Among them we noticed that software-based Page Coloring, hardware-based CAT, and ARM Autolock and Cache Bar are effective techniques in combating these attacks.

6. OPEN RESEARCH QUESTIONS

Even though in this survey we tried to analyze the most common side channel attacks based on the latest research works, there are still a lot of open questions regarding how to manage such attacks. Most of the recent works are based on attack detection, on the other hand, the management of its negative side effects are remaining as points of concern. One of the interesting points that have been left unexplored is in the category of security policies which should be taken upon side-channel attack detection. This concept means what actions should be taken after the detection of an attack, or how much the false detection rate is, and the system should manage the false detection.

The second category that we can think of is hardware level solutions that are unexplored, e.g. new secure and practical last-level cache inclusiveness, better design of cache coherence protocols, and effect of cache invalidation instructions.

The third category is system level solutions. Solutions in cloud computing has been broadly studied, yet the solutions that work well with the cloud business model are still unexplored. Moreover, scenarios like mobile OS or browsers are less explored which can include cache partition for JavaScript code or Android-level cache partitioning model.

7. REFERENCES

[1] F. Liu, Y. Yarom, Q. Ge, G. Heiser and R. B. Lee, *Last Level Cache Side Channel Attacks are Practical* S&P ′15.

[2] G. Irazoqui, T. Eisenbarth and B. Sunar, *A Shared Cache Attack That Works across Cores and Defies VM Sandboxing – and Its Application to AES* S&P ′15.

[3] Y. Yarom and K. Falkner, *FLUSH+RELOAD: a High Resolution, Low Noise, L3 Cache Side-Channel Attack* USENIX Security ′14.

[4] D. Gruss, C. Mauricet, K. Wagner and S. Mangard, *Flush+Flush: A Fast and Stealthy Cache Attack* DIMVA ′16.

[5] C. Disselkoen, D. Kohlbrenner, L. Porter and D. Tullsen, *Prime+Abort: A Timer-Free High-Precision L3 Cache Attack using Intel TSX* USENIX Security ′17.

[6] S. Schaik, C. Giuffrida, H. Bos, and K. Razavi, *Malicious Management Unit: Why Stopping Cache Attacks in Software is Harder Than You Think* USENIX Security ′18.

[7] Y. Ye, R. West, Z. Cheng and Y. Li, *COLORIS: A dynamic cache partitioning system using page coloring* PACT ′14.

[8] F. Liu, Q. Ge, Y. Yarom,F. Mckeen, C. Rozas, G. Heiser and R. Lee, *CATalyst: Defeating Last-Level Cache Side Channel Attacks in Cloud Computing* HPCA ′16.

[9] M. Green, L. R. Lima, A. Zankl, G. Irazoqui, J. Heyszl, and T. Eisenbarth, *AutoLock: Why Cache Attacks on ARM Are Harder Than You Think* USENIX Security ′17.

[10] Z.Zhou, M.Reiter, Y. Zhang *A Software Approach to Defeating Side Channels in Last-Level Caches* CCS ′16.

[11] Application Note by Intel, *TLBs, Paging-Structure Caches, and Their Invalidation*

[12] A. Silberschatz, P. B. Galvin, and G. Gagne, *OPERATING SYSTEM CONCEPTS*, John Wiley Sons, 9th Edition, 2013.

[13] C. Jin, *Side Channel Attacks*, University of Connecticut, Spring 2016, www.scl.uconn.edu/courses/ece6095/lectures/sidechannels.pdf.

[14] T. W. Barr, A. L. Cox, and S. Rixner, *Translation Caching: Skip, Don't Walk (the Page Table)* ISCA $'$10.

[15] A Note by J. Rosenberg on Science Direct, *Learn more about Side Channel Attack*, 2017

[16] G. Irazoqui, and X. Gou, *LLC Attacks Applicability and Countermeasures* Enigma$'$17