Exploiting Secrets by Leveraging Dynamic Cache Partitioning of Last Level Cache

Anurag Agarwal
Department of CSE
Indian Institute of Technology Ropar, Punjab, India.
2018csm1006@iitrpr.ac.in

Jaspinder Kaur
Department of CSE
Indian Institute of Technology Ropar, Punjab, India.
2017csz0002@iitrpr.ac.in

Shirshendu Das
Department of CSE
Indian Institute of Technology Ropar, Punjab, India.
shirshendu@iitrpr.ac.in

***Abstract*—Dynamic cache partitioning for shared Last Level Caches (LLC) is deployed in most modern multicore systems to achieve process isolation and fairness among the applications and avoid security threats. Since LLC has visibility of all cache blocks requested by several applications running on a multicore system, a malicious application can potentially threaten the system that can leverage the dynamic partitioning schemes applied to the LLCs by creating a timing-based covert channel attack. We call it as *Cache Partitioned Covert Channel* (CPCC) attack. The malicious applications may contain a trojan and a spy and use the underlying shared memory to create the attack. Through this attack, secret pieces of information like encryption keys or any secret information can be transmitted between the intended parties. We have observed that CPCC can target single or multiple cache sets to achieve a higher transmission rate with a maximum error rate of 5% only. The paper also addresses a few defense strategies that can avoid such cache partitioning based covert channel attacks.**

***Index Terms*—Last Level Cache, Covert Channel Attack, Cache Partition, Timing Channel Attack**

I. Introduction

The present-day System on Chip (SoC) contains multiple cores integrated on a single chip, thereby allowing concurrent execution of instructions [1]. However, the applications running on various cores use shared hardware resources such as LLC, DRAM, and underlying communication network that may create a logical communication channel between two malicious applications. The malicious application uses the logical channel to leak sensitive information, thereby hampering the security aspects of such SoC platforms [2]–[5].

Threat Model: The threat model used in this paper considers a multicore system where each core has a private L1 cache, and an inclusive LLC shared among all the other cores. To obtain fairness among the applications, LLC is dynamically partitioned among them as per their demands. The cores are connected using a bus-based interconnect. When the processors request cache blocks, a core running a malicious application (Trojan) can send secret information to another malicious application running on a different core (Spy). Thus, the threat model assumes that at least two cores host malicious applications while the rest of the applications are secured/trusted. We also assume that the trojan has somehow retrieved some secret information such as an encryption key or a piece of sensitive information that it wants to send to the spy as binary bits but it cannot reveal the information directly as it may get identified [6]–[8]. So the trojan uses the shared hardware (LLC) to send the information as binary bits to the spy, which is outside the secured enclave using a covert channel [9].

To create a covert channel attack, two properties are must: (a) creating a channel through which information can be leaked, and (b) modulating the channel for communication between trojan an spy. The paper shows that dynamic cache partitioning [10]–[14] can be used to establish a covert channel between trojan and spy called as Cache Partitioned Covert Channel (CPCC). It requires a shared and dynamically partitioned LLC, a number of memory addresses mapping to pre-decided sets, and fine-grained latency measurement for measuring memory accesses latencies. Memory addresses used by an attacker can be found by performing reverse engineering on the system under attack. While latency can be measured using some assembly code instructions such as *mov* (for accessing a memory address) and *rdtscp* (reads current value of time-stamp counter) [15]. When a *mov* instruction is surrounded by *rdtscp* instruction, the time difference in time-stamp is the measured latency for that memory access.

Dynamic cache partitioning is vulnerable to security attacks as most of the cache partitioning schemes monitor only a few pre-determined cache sets for an application's usage [10]–[14]. Based on the usage, the rest of the cache is partitioned among the applications. Thus, the trojan and spy can target the pre-determined sets to launch the CPCC attack. We evaluate CPCC on numerous dynamic cache partitioning methods [10]–[14] to show that the existing techniques are prone to the attack. To the best of our knowledge, none of the works explored the vulnerability that might occur due to the dynamic cache partitioning of shared LLC.

The major contributions of the paper are:

1) We show a novel approach by which a timing-based covert channel can be created on a dynamically partitioned shared LLC. We call it as CPCC.
2) We show a variety of partitioning schemes that are found to be vulnerable to such attacks (CPCC).
3) We put forward some defense mechanisms that can prevent CPCC attack on shared LLC.

The rest of the paper is organised as follows. Background and related works are discussed in section II. Section III describes

978-3-9819263-5-4/DATE21/©2021 EDAA

CPCC attack model. Experimental analysis are shown in section IV. Section V discuss some defense mechanism against CPCC attack and Section VI concludes the paper.

II. Background and Related Work

A covert channel is an intended channel formed for transmission of secret information between *trojan* and *spy* [9]. Trojan and spy are independent processes that runs on different cores where the trojan sends secret information to the spy through the channel. Among various covert channel attacks, timing-based channels are popular where information is transmitted when there exists a noticeable timing variation among the operations. One such timing-based covert channel attack can be created due to shared LLC present in modern processors [15]. The possibility of such attack may originate during the manipulation of cache blocks states such as coherence, LRU bits (during cache hit/misses) which may result in difference in the block access latency. The latency difference may allow the parties to convey bit by bit information. Two well-known techniques to measure the timing differences are *Prime+Probe* [6] and *Flush+Reload* [15].

Cache partitioning schemes [10]–[14] are used to partition the shared LLC among multiple cores based on application demand. Qureshi et al. proposed Utility-based Cache Partitioning (UCP) [10] that tracks the utility information of cache sets using a circuit, Utility Monitor (UMON) to partition the cache. UMON consists of Auxiliary Tag Directory (ATD) and hit counters. A hit in ATD at recency position p states that if a number of ways allocated to a core are greater than or equal to p, it will result in a cache hit otherwise a miss. On a cache hit, the hit counter is incremented and depending on the hit counters, UCP partitions the cache that can facilitate maximum hits. Each core has separate ATD in the LLC. To reduce the hardware overhead, ATD is applied to a limited number of finalized sets according to Dynamic Set Sampling (DSS). Such sets are called as DSS-sets. The results obtained from these sets are applied to all the LLC sets to partition the cache.

Xie et al. proposed a pseudo partitioning scheme PIPP [11] that leverages the fact that the majority of blocks within a cache tend to be dead-blocks. Halwe et al. proposed CCPIP [12] that removes the problem of cache stealing from PIPP by allowing cores to steal/donate only up to a certain threshold, thereby preventing to steal all cache ways as in PIPP. Yang et al. proposed CIACP [13] used on Coarse-Grained Reconfigurable Array (CGRAs) systems only. It uses a UMON, correlation monitoring, and iteration monitoring circuit for each CGRA. UMON monitors the utility information of each CGRA; the correlation monitor accounts for the overlapping data between CGRAs, and the iteration monitor counts the iteration number of each CGRA. Depending on all the counters, an effective partition is determined. Newton et al. proposed DAAIP [14] that uses the variations in dead-block count for different application phases. It determines an application phase as either less-lively or lively phase based on the percentage of dead-block present in a phase. A block is inserted at LRU position in less-lively phase whereas, in lively phase, a block is inserted at (LRU-1) position. However, DAAIP is not an explicit partitioning scheme as it indirectly creates a partition by changing the replacement policy. In the literature, dynamic partition only concentrates on the performance evaluation of the processes running in the cores.

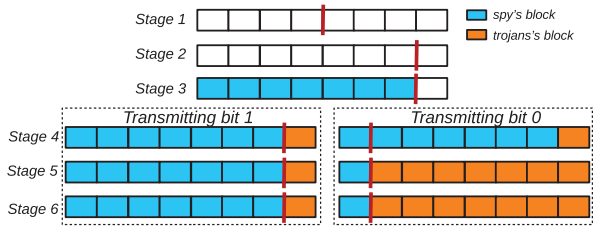

Fig. 1. Cache partitioning covert channel attack where bit 1 (left figure) and bit 0 (right figure) is transmitted from trojan to spy using UCP [10].

III. Proposed Technique: Cache Partitioned Covert Channel (CPCC)

In LLC's dynamic partitioning case, a covert channel is possible to establish because the applications indirectly control the LLC partition size. As per the application's memory requirement, the cache is partitioned, making the system vulnerable to CPCC attacks. The paper initially explains CPCC using UCP [10] as the cache partition method. Section IV further describes the possibility of CPCC attacks in various partitioning methods [11]–[14].

A. Overview of CPCC:

Let us assume a system with two cores: core-0 and core-1 with an 8 way set-associative shared LLC and uses UCP [10] as the underlying cache partitioning scheme. Core-0 is secured using some security protocols and the application running in the core is an important application such as secured block cipher, AES [6]. Suppose, core-0 hosts a trojan that wants to share some secret information such as the AES key (0's and 1's) to a spy residing in a non-secured zone at core-1. Since trojan is in a secured zone, it cannot send any secret information directly to the spy. Figure 1 shows a particular set, s, also known as *target-set* in the shared LLC that is used by the trojan and spy to transmit secret information in binary bits. The set s is the DSS-set that is used by UCP to monitor the cache usage. For the ease of understanding and fair distribution of ways among the applications, we assume that initially both the application equally distributes the ways as shown in Stage-1 of the Figure 1.

Launching a CPCC attack: Let $addr_1$ and $addr_2$ be two sets of unique addresses used by the spy and trojan, respectively. All the addresses in these two sets are mapped to a DSS-set. To facilitate a partition change, the minimum count of addresses in $addr_1$ and $addr_2$ is seven each (as each set has 8-ways). In the figure, blue and orange colors are the blocks of spy and trojan, respectively. Initially, the spy sequentially accesses (multiple times) all the blocks from $addr_1$. Since the LLC working set available to the spy is 4, each access results

in a cache miss. After a few execution cycles, UCP understands using its UMON's circuit that assigning 7 ways to the spy may result in a massive reduction in the cache miss while incurring no loss to the other process. Therefore, UCP allocates seven ways to the spy and only one way to the trojan, as shown in Stage-2 of the figure. Though the UCP monitors only DSS-sets, the partition applies to all the cache sets. The target-set shown in the figure need not be a DSS-set. Now, the spy fills the target-set with some pre-decided blocks, as shown in Stage-3.

Transmitting bit 1: In the figure, the left half indicates that bit 1 is transmitted and the right half indicates the transmission of a bit 0 by the trojan to the spy. We initially describe the transmission of a bit 1 from Stage-4 (left). To transmit bit 1, the trojan does not perform any activity on the target-set to replace the spy's blocks. Also, trojan does not try to change the partition. In Stage-4, it can be observed that trojan only requested for one block within its allocated partition. In the next step, the spy re-accesses its pre-decided memory blocks and measures the memory access latency. Since all the access results in a cache hit, the spy deciphers that bit 1 is transmitted by the trojan.

Transmitting bit 0: To transmit bit 0, the trojan uses the same logic of sequentially accessing its blocks from $addr_2$ to perform a partition change. Since all the block accesses results in cache miss in the DSS-sets, UCP changes the cache partition by allocating 7-ways to the trojan and 1 to the spy as shown in Stage-4 (right). In Stage-5, the trojan brings its blocks and places in the target-set. This results in replacing spy's blocks. When the spy accesses its pre-decided memory blocks, it finds an increase in the memory access latency. From this, the spy deciphers that bit 0 is transmitted by the trojan. Thus, the secret key, which is a series of 0's and 1's, is indirectly transmitted to the spy using CPCC.

The only requirement for CPCC attack in UCP based partitioning method is the knowledge of DSS-sets. Since a partition change in UCP occurs by monitoring only the DSS-sets, the trojan and spy must have information of such sets beforehand to change the partitions dynamically.

B. Generic Algorithm for CPCC attack

Algorithm 1 explains a generic algorithm proposed to create a CPCC attack to transmit 1-bit of information from trojan to spy. Repeated use of the same algorithm can transmit the complete message. Some abbreviations used in the algorithm is defined below.

- **Target-set (s):** Both trojan and spy agrees on the set used for transferring the information known as target-set. Multiple target-sets can be used in parallel to increase the transfer rate.
- **Partition size (w):** This parameter is used to modulate the maximum partition size allocated to each process while executing the attack.
- **Memory addresses:** Four sets of memory addresses namely: $addr_1, addr_2, p_1$, and p_2 are required. The size of $addr_1$ and $addr_2$ varies depending on the partitioning algorithm. Blocks in $addr_1$ and $addr_2$ are mapped to the DSS-sets and they are used to facilitate the change in LLC partition. The addresses in p_1 (prime addresses) and p_2 (probe addresses) should map to the target-set and size of both p_1 and p_2 is w. All the addresses can be disjoint and there is no need for a shared library.

Algorithm 1: Cache Partition channel

```
Function partitionChange(m):
    Access m blocks in some order to enforce partition change
Function Spy(addr1, p1):
    partitionChange(addr1) // Stage 1
    sleep;
    // Stage 3
    for i ← 1 to w do
        Access block p1[i];
    sleep;
    // Stage 5 & 6
    Measure time while accessing block p1[i] and check whether hit/miss;
    if hits > misses then
        bit = 1;
    else
        bit = 0;
    return bit
Function Trojan(addr2, p2, bit):
    sleep;
    // Stage 4
    if bit = 0 then
        partitionChange(addr2)
        for i ← 1 to w do
            Access block p2[i];
    sleep;
```

The attack model presented in this paper is a novel attack that is different from the Prime+Probe attack [6] as CPCC requires six stages to launch the attack, unlike three stages in Prime+Probe. The six stages of launching a CPCC attack are:

Stage 1: The spy enforces the partitioning scheme to allocate w ways to it by fetching blocks from $addr_1$ in some order depending on the partitioning scheme. The partition changing policy may vary in different partitioning algorithm but is almost same in most of the UMON based partitioning schemes like UCP [10], PIPP [11], CCPIP [12], etc.

Stage 2: The partition manager computes the sub-optimal partition between the cores depending on the memory access in the past. As each of the previous spy's memory access is a miss and the total block accessed is w, the partition manager allocates w ways to the spy in the view that its future performance may improve provided that the performance of other applications are not hampered.

Stage 3: After partition change, the spy fills (or primes) the target-set with known cache blocks from p_1 to make target-set in a partially known state.

Stage 4: This stage decides the bit to be transmitted by the trojan to the spy. The trojan either forces the partition to re-adjust and replace some of the spy's cache blocks (bit 0), or it

does not replace any of the spy's block (bit 1).

Stage 5: Spy accesses its blocks (p_1), which may result in a cache hit or a miss that is determined by the previous step (trojan either replaced spy's block or not).

Stage 6: Based on Stage 5, spy determines the bit by measuring the memory access time. Since a cache miss requires more time to access than a cache hit, spy monitors the timing difference and also the count of misses/hits. Based on the information collected, the spy decodes the information being sent by the trojan.

Algorithm 1 is a general template that requires fine-tuning of some parameters depending on the underlying cache partition employed and other performance units within the system under attack such as timing of each stages, hit/miss latency threshold, number of cache ways (w), memory address sets: $\{addr_1, addr_2, p_1, p_2\}$, and the $partitionChange$ method. Due to cache partitioning schemes' varying nature, the $partitionChange$ method can be employed to access memory addresses in sequential, random, or a specific order. The generic algorithm discussed above uses only one target-set to transmit information. But the algorithm can be easily extended for multiple sets trivially as explained in the next subsection.

C. Multi-Set Generic Algorithm

In a dynamic cache partitioning scheme, the utility of one set affects the utility of the other sets in the cache. Due to this nature, our approach, as described in Algorithm 1, can be easily modified for a possible CPCC attack through multiple sets. Let n be the number of sets used to transfer information by the trojan to the spy. This requires n memory address pairs namely $\{(p_{11}, p_{21}), (p_{12}, p_{22}), \ldots, (p_{1n}, p_{2n})\}$ mapping to the target-sets $\{s_1, s_2, \ldots, s_n\}$ respectively.

The multi-set CPCC attack requires an equal number of steps as that of the single-set discussed in the previous subsection. In the first stage, the spy enforces partition change using $addr_1$ and the DSS-sets. At stage 2, the underlying partitioning scheme changes the partition in spy's favor assigning w ways to all cache sets (due to way based cache partitioning). In the next stage i.e., Stage 3, spy primes all the cache sets $\{s_1, s_2, ..., s_n\}$ using the addresses from the set $\{p_{11}, p_{12}..., p_{1n}\}$. Stage 4 is similar to that of the stage 4 of Algorithm 1. In stage 5, the trojan transfers a vector of binary bits V where $|V| = n$. Depending on the bit V_i, trojan either access or does not access the cache set s_i. At the final stage, i.e., stage 6, spy probes the cache sets used in stage 3 and determines the number of misses and hits using memory latencies individually for each set. Information can be retrieved by the same order used by trojan in the previous stage i.e. s_i used for the i^{th} bit where $1 \leq i \leq n$.

IV. Experimental Analysis

We experimented CPCC attack with various cache partitioning techniques: UCP [10], PIPP [11], CCPIP [12], CIACP [13], and DAAIP [14] using an in-house multi-core trace-based simulator, *ZeusSim*. Some fine-tuning in the synchronization mechanism, hit-miss threshold, and traces are made to create a covert channel, as discussed in subsection IV-A. The simulator takes a machine configuration and memory trace for each core as input. Table I contains the machine configurations used for our experimental analysis. The system consists of a dual-core processor with two levels of caches L1 (private) and L2 (shared) where L2 is dynamically partitioned and used as LLC. We also discuss the fine-tuning required for the cache partitioning technique. ZeusSim can be easily extended for various cache partitioning schemes, replacement policies, and coherence policies as per the users' choice for an extensive experimental evaluation. Since this paper aims to show the possibility of a CPCC attack, we have used synthetic traffic to analyze the system's performance and energy consumption.

TABLE I
Simulation Parameters

Component	Configuration
Processors	Dual-Core, 3 GHz.
L1 Cache	64KB, 4-way, 64B line-size, private
L2 Cache	2MB, 16-way, 64B line-size, shared LLC
DRAM Size	2 GB

A. Changes made in various cache partitioning techniques to create a covert channel for CPCC attack

In UCP, we have used 5 million cycles to make changes in memory partition using UMON. Initially, UCP allocated equal ways to each core, i.e., each trojan and spy is assigned eight ways each. For the first 5 million cycles, spy repeatedly accesses blocks from $addr_1$. UCP monitors the cache usage and allocates 15 ways to spy to reduce the miss rate. For the next 5 million cycles, spy primes the cache set only once, then becomes idle. In the meantime, trojan decides to send either bit 0 or 1 and it manipulates the cache set according to its need. In the last 5 million cycles, the spy probes the cache with blocks from p_1 and decodes the information. It uses both single-set and multi-set algorithm to create a covert channel. Both PIPP and CCPIP internally follow UMON as proposed in UCP for the cache partitioning. Hence implementing them in *ZeusSim* is almost same as UCP. Since both PIPP and CCPIP allow to deviate from the partition set by the UCP, the attack may perform without changing the partitions mentioned in Algorithm 1. But for this experiment, we have used the approach mentioned in Algorithm 1.

CIACP introduces the concept of overlapping data and iteration number to decide the memory partition along with the UMON circuit. As in our attack, there is no overlapping (or shared) memory block; therefore, the impact of overlapping data is nullified. The application always dominates the number of iterations that forces for a partition change, i.e., whenever the spy enforces partition change, the trojan is idle and vice-versa. Due to this, the impact of the iteration number is significantly low to create a covert channel and the attack method used in CIACP is similar to that of UCP.

Unlike the previous cache partitioning schemes, DAAIP is not an explicit partitioning technique, as mentioned in Section II. Thus, the partition change stages are not required for

TABLE II
ERROR RATE OF CACHE PARTITIONING SCHEMES UNDER CPCC

Cache Partitioning	% Error Rate	
	Single Target-Set	Multiple Target-Sets
UCP	0.74	0
PIPP	0.18	0
CCPIP	5.56	0
CIACP	1.28	0
DAAIP	0	0.58

DAAIP. Initially, the spy primes the cache set by accessing each memory block twice in order to promote the block from the LRU position to a higher position in the replacement stack. The two accesses are done to avoid replacing the block from the cache. The trojan then accesses or does not access the block depending on the bit to be transmitted. The spy later probes the cache block and deciphers the information.

B. Results

Various parameters are evaluated, such as the error rate of the transmitted information and spy's observation latency. Memory addresses for sets $addr_1$, $addr_2$, p_1 and p_2 are generated such that each address is unique. Also, blocks from addresses $addr_1$ and $addr_2$ maps to the DSS-sets whereas, block from addresses p_1 and p_2 maps to the target-set. Due to cache partitioning schemes' varying nature, the transmission rate heavily depends on the underlying partitioning scheme. For example, in UCP, the partition changes in every 5 million cycle, and the single-set algorithm requires about 10 million cycles to send a single bit of information (stage 5 requires a partition change accounting for 5 million cycles + for transferring bits by the trojan another 5 million cycles is required during which the spy waits). As CPCC requires two partition changes, the transmission rate is $\approx n/(2*t)$ bits per cycle where n is the number of target-sets used, and t is the interval (in cycles) for a partition change. As partition is changed in millions of cycles, the transmission rate of CPCC attack having only a single target-set is slow.

The error rate is the ratio of the number of bits resulting in error to the total number of bits transferred. The experiments performed uses a message size of 5000 bits transferred between the trojan and spy to evaluate CPCC attack using both single target-set and multiple target-sets. The error rate measured is calculated as an average of ten simulations. In multiple target-set experiments, a total of 64 target-sets are used. Table II shows the error rate observed in different partitioning schemes. When a single target-set is used, UCP resulted in less than 3% error rate while transmitting the message. PIPP, due to its inherent use of UCP, shows a comparable error rate. While in CCPIP, the error rate observed is more because it uses threshold values that sometimes prevent the partitioning scheme from allocating w ways to spy or trojan. This leads to some misinterpretation of bits, thereby resulting in more errors. CIACP, due to its negligible impact of the iteration number, prevents appropriate partition size and increases the error rate by about 0.5% compared to UCP. Since DAAIP does not enforce an explicit partition, it leads to a very minimal error. But when multiple target-sets are used, the error rate reduces to almost 0% for all the partitioning schemes. This is due to the requirement of fewer partition changes for the same number of bits than a single target-set.

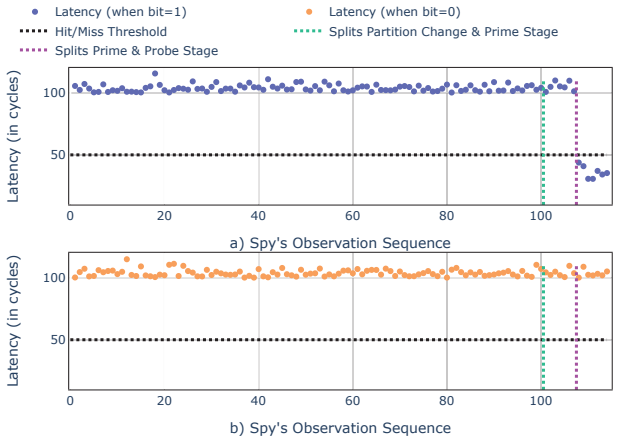

Fig. 2. Example sequence of spy latency when (a) bit = 1 and (b) bit = 0 is transferred using UCP. Blue and orange dots denote latencies observed by the spy, black line denotes hit/miss latency threshold used for L2 cache, and green & violet line divides stage 1 of algorithm 1 into three parts.

Figure 2 shows the latency observed during memory accesses performed by a spy during a CPCC attack. The plot shows observed latencies when the bit transmitted is 1 (top) or 0 (bottom). Time intervals when the spy is idle are ignored and reduced the number of latencies for easy readability. The black line denotes the hit/miss latency threshold (latency below this threshold constitutes a cache hit otherwise a cache miss) used for L2 cache, and the plot shows three stages of the CPCC attack. The hit/miss threshold is set to 50 cycles for this experiment. The first stage (from 1st memory access to green line) shows stage 1 of CPCC attack where spy enforces partition change by accessing memory blocks from $addr_1$. As the memory access initially results in a cache miss, it shows a higher latency of about 100 cycles. The second phase (between green and violet line) shows stage 3, where the spy primes the target-set using the addresses in p_1. As the addresses are not previously cached, they result in cache misses. The final phase (from the violet line) shows the probe phase where spy probes the cache set and determines the number of hits and misses. When the bit transmitted is 1, the trojan does not manipulate the target-set. The spy, upon probing the set, results in cache hits (45 cycles latency). Similarly, when the bit transmitted is 0, trojan manipulates the target-set resulting in the eviction of some p_1 blocks. Upon probing the set by the spy, it results in cache misses. Hence a higher latency of 100 cycles is obtained.

V. DEFENSE STRATEGIES

In this section, we have discussed two possible approaches to defend a CPCC attack, but both have a negative impact on the system's performance. Therefore, proposing an efficient countermeasure for the CPCC attack remains an open challenge.

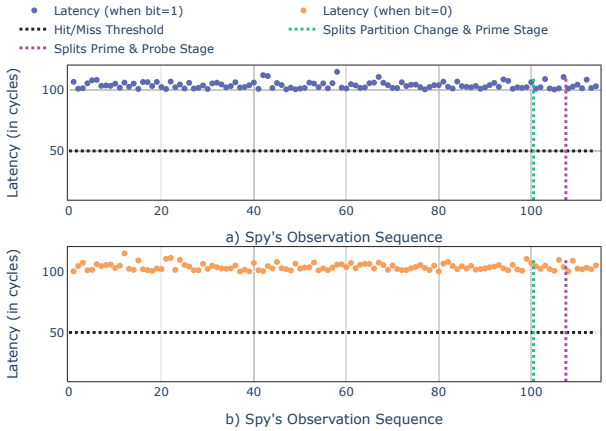

Fig. 3. Example sequence of Spy's memory access latency when a) bit = 1 and b) bit = 0 is transferred after the Invalidation defense measure is applied.

TABLE III
ERROR RATE OF CACHE PARTITIONING SCHEMES UNDER CPCC WHEN RANDOMNESS IN TIMING PARTITION CHANGE ADDED

Cache Partitioning	% Error Rate	
	Single Target Set	Multiple Target Sets
UCP	50.70	47.18
PIPP	49.57	29.70
CCPIP	47.98	30.81
CIACP	48.71	44.49
DAAIP	0	0.58

A. Invalidation

While changing the cache partition, the cache controller can invalidate the lines whose association changes. Thus, whether accessed or not, all the lines will result in cache misses, confusing the spy to differentiate between bit 0 and 1. Spy's memory latency plot after modifying the cache partitioning scheme is shown in Figure 3. As shown, all memory accesses are above the hit/miss threshold resulting in a cache miss, due to which the spy interprets all the bits as 0. This results in an error rate of 50%. To improve the system's performance, manufacturers, instead of invalidating all the cache blocks participated in partition change, may randomly invalidate some of the blocks, making it hard for the spy to interpret the bit.

B. Randomization

The majority of dynamic cache partitioning scheme in existence uses some mathematical logic to find a sub-optimal partition. After gathering enough information about the logic, it is easy to fine-tune the presented model to create a cache partition covert channel. To prevent processes from creating a covert channel, randomness can be added in any form like time to change partition (both processes will become unsynchronized) or change mapping function randomly (may result in performance degradation) such that the processes do not map to the same target set. Randomness in time to change partition is evaluated and found to be effective. The trojan and spy being out of synchronization, the bit communicated, are no better than a random guess. Table III shows that all the partitioning scheme shows an error rate of about 50% when single target-set is used, which can also be attained by random guessing. When multiple target-sets are used, UCP and CIACP show an error rate of 47% and 44%, respectively. In PIPP and CCPIP, the pseudo partitioning scheme increases the error rate by 30%.

VI. CONCLUSION

The paper presents the vulnerabilities of various cache partitioning schemes. As the applications directly control the dynamic change in memory partition, any application that has a malicious intention can exploit the underlying dynamic cache partitioning scheme to create a CPCC attack. The channel is used to transfer secret information between the intended malicious parties, thereby possessing security threats to the system. We have proposed an attack model and evaluated various partitioning schemes that show the possibility of a CPCC attack on a system that has shared caches. We have measured that the error rates in information transmission in these partitioning schemes tested are below 5% when a single target-set is used. But the same error rate dropped to nearly 0% when multiple target-sets are used. This shows the possibility of a CPCC attack in a modern multicore system. We have also proposed some defense mechanisms against the CPCC attack, but such a mechanism negatively impacts system performance. As future work, we focus on proposing an efficient countermeasure to prevent the CPCC attack without degrading the system performance.

REFERENCES

[1] W. Aspray, "The intel 4004 microprocessor: what constituted invention?" *IEEE Annals of the History of Computing*, vol. 19, no. 3, pp. 4–15, 1997.
[2] G. Dessouky and A. S. T. Frassetto, "HybCache: Hybrid Side-Channel-Resilient Caches for Trusted Execution Environments," in *USENIX Security Symp.*, 2020.
[3] G. Daniel, R. Spreitzer, and S. Mangard, "Cache Template Attacks: Automating Attacks on Inclusive Last-Level Caches," in *USENIX*, 2015, p. 897–912.
[4] F. Farahmandi, Y. Huang, and P. Mishra, *System-on-Chip Security: Validation and Verification*, 01 2020.
[5] M. Lipp *et al.*, "Meltdown: Reading Kernel Memory from User Space," in *USENIX*, 2018.
[6] D. A. Osvik, A. Shamir, and E. Tromer, "Cache Attacks and Countermeasures: The Case of AES," in *Topics in Cryptology – CT-RSA 2006*, D. Pointcheval, Ed.
[7] Z. H. Jiang and Y. Fei, "A novel cache bank timing attack," in *ICCAD*, 2017, pp. 139–146.
[8] D. J. Bernstein, "Cache-timing attacks on AES," 2005.
[9] B. W. Lampson, "A Note on the Confinement Problem," *Commun. ACM*, vol. 16, no. 10, p. 613–615, 1973.
[10] M. K. Qureshi and Y. N. Patt, "Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches," in *MICRO*, 2006, pp. 423–432.
[11] Y. Xie and G. H. Loh, "PIPP: Promotion/Insertion Pseudo-Partitioning of Multi-Core Shared Caches," in *ISCA*, 2009, p. 174–183.
[12] P. D. Halwe, S. Das, and H. K. Kapoor, "Towards a Better Cache Utilization Using Controlled Cache Partitioning," in *DASC*, 2013, pp. 179–186.
[13] C. Yang, L. Liu, K. Luo, S. Yin, and S. Wei, "CIACP: A Correlation- and Iteration- Aware Cache Partitioning Mechanism to Improve Performance of Multiple Coarse-Grained Reconfigurable Arrays," *TPDS*, vol. 28, no. 1, pp. 29–43, 2017.
[14] Newton, S. K. Mahto, S. Pai, and V. Singh, "DAAIP: Deadblock Aware Adaptive Insertion Policy for High Performance Caching," in *ICCD*, 2017, pp. 345–352.
[15] Y. Yarom and K. Falkner, "FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack," in *USENIX*, 2014, pp. 719–732.