Cache Bandwidth Contention Leaks Secrets

Han Wang, Ming Tang*, Ke Xu, Quancheng Wang
Key Laboratory of Aerospace Information Security and Trusted Computing,
Ministry of Education, School of Cyber Science and Engineering
Wuhan University
Wuhan, China
Email: {han.wang, m.tang, kexuwhu, wangquancheng}@whu.edu.cn

***Abstract*—In the modern CPU architecture, enhancements such as the Line Fill Buffer (LFB) and Super Queue (SQ), which are designed to track pending cache requests, have significantly boosted performance. To exploit this structure, we deliberately engineered blockages in the L2 to L1d route by controlling LFB conflict and triggering prefetch prediction failures, while consciously dismissing other plausible influencing factors. This approach was subsequently extended to the L3 to L2 and L2 to L1i pathways, resulting in three potent covert channels, termed L2CC, L3CC, and LiCC, with capacities of 10.02 Mbps, 10.37 Mbps, and 1.83 Mbps, respectively. Strikingly, the capacities of L2CC and L3CC surpass those of earlier non-shared-memory-based covert channels, reaching a level comparable to their shared memory-dependent equivalents. Leveraging this congestion further facilitated the extraction of key bits from RSA and EdDSA designs. Coupled with SpectreV1 and V2, our covert channels effectively evade the majority of traditional Spectre defenses. Their confluence with Branch Prediction (BP) Timing assaults additionally undercuts balanced branch protections, hence broadening their capability to infiltrate a wide range of cryptography libraries.**

***Index Terms*—covert channel, side channel, CPU microarchitecture**

I. Introduction

Modern processors use complex optimization structures to boost performance, introducing potential security risks. Early research [1], [2] focused on exploiting cache structures to launch attacks on cryptography libraries. Recent investigations [3]–[7] have concentrated on compromising process isolation. Notably, the Spectre [6] and Meltdown [5] attacks demonstrated that even uncommitted secrets could be exposed. These groundbreaking discoveries have drawn considerable attention as they underscore a disconcerting possibility: flaws inherent in hardware could result in confidential information leaking from reliable to malicious processes, even in the absence of any software vulnerabilities. A novel microarchitecture attack often relies on unexplored elements of the microarchitecture or leverages these elements in an entirely new way.

Lord of the Ring(s) [3] was the first to reverse engineer the CPU's ring interconnect, discovering that data packets from different CPU cores could compete on the ring interconnect, resulting in time delays. Subsequent work [12] has expanded upon this base, further reverse engineering and attacking server CPU mesh-connect structures. Another notable work [10], focuses on the contention issue in the transmission lines between a core's frontend and execution engine.

The discussion of transmission lines does not stop here. Following the issuance of a load or store instruction, this instruction might sequentially access the L1, L2, and L3 caches. This pathway has not been seen as a bottleneck in execution time according to Intel manuals [13]. However, this viewpoint fails to consider the situations of Line Fill Buffer (LFB) saturation and hardware prefetching. LFB is a buffer utilized by the CPU to track and optimize pending memory requests, such as combining multiple stores or reads. By creating contention for LFB and prefetching prediction failures, we successfully induced congestion in the cache bandwidth. Moreover, we have eliminated the influence of factors like execution port contention and L1 cache misses through experiments. Based on these findings, we extended our analysis, unearthing similar contention on the routes from L2 to L1 instruction cache (L1i) and from L3 to L2.

This paper makes the following contributions:

- We present the first study focusing on cache bandwidth contention, and through reverse engineering, provide an extensive discussion on the causes of cache bandwidth congestion.
- We have constructed three high-speed and noise-resistant covert channels, relying on cache bandwidth contention. Our covert channels deliver capacities up to 10.024 Mbps (L2CC), 10.369 Mbps (L3CC), and 1.838 Mbps (LiCC) from a single thread. L2CC and L3CC are, to the best of our knowledge, the fastest microarchitectural covert channels that do not depend on shared memory.
- We leverage cache bandwidth contention to leak key bits from RSA and EdDSA. We have improved existing Spectre attacks and BP Timing attacks using our covert channels to bypass several defenses and enhancing the attack potential.

II. Related Works

Speculative execution attacks, a notable exemplar being the Spectre-V1 attack [6], transpire when an attacker manipulates the branch predictor, leading to incorrect execution of instructions by the victim. In such instances, despite the failure of array length checks, the CPU can be misled into speculatively

*Corresponding author

979-8-3503-4859-0/DATE24/© 2024 EDAA

executing out-of-bounds array accesses. Similar attacks are also witnessed in the context of indirect branches.

LFB-related attacks. Previous LFB-related attacks were primarily focused on leaking information directly from the LFB. This flaw was initially exposed by RIDL [15], in which the CPU speculatively loaded a value from an LFB that belonged to a different process. As prior LFB-related attacks could not leak data arbitrarily, CacheOut [16] introduced a new cache pathway to ensure that attackers would not have to wait for information in the LFB to become available, but could instead actively move it to the LFB using cache eviction. Our attack, unlike previous ones, does not directly leak data from LFB or Super Queue but utilizes contention to bypass potential defenses.

Transmission-lines related attacks. Lord of the Ring(s) [3] attack established a cross-core covert channel based on contention by reverse-engineering the protocol of ring interconnects. They also exploited the competition within ring interconnects to infer the victim's secret. Subsequent work [12] has further examined and attacked the mesh structure in server CPUs. There are other studies emphasizing SOC buses, as highlighted in [18] and [19]. These articles provide valuable insights into the subject matter. Frontend Bus [10] determined the shared strategy of the front-to-back transmission bus within a physical core by analyzing the effects of different loops on the execution time of the receiver. They proposed two new covert channels based on this discovery, but they did not test the covert channels in a noisy environment. Our covert channels also utilize transmission lines. However, we ensure superior transmission bandwidth and have conducted evaluations in noisy conditions.

Cache defenses against speculative execution attacks. Based on the classification in [17], defenses can be classified into No setup, No Access without Authorization, No Use without Authorization, and No Send without Authorization. The innovations presented in section V-C demonstrate that combining with our covert channels, Spectre can surpass some defenses, excluding No Access without Authorization (CSF-LFENCE [8]) and Delay in No Send without Authorization (DOLMA [9]). These two categories have a huge performance overhead.

III. Reverse Engineering

Unless bearing an uncacheable tag, a read or write instruction may sequentially access the L1, L2, and L3 caches, contingent on the storage status. This data path has not been seen as a bottleneck in terms of execution time. Nonetheless, we have managed to induce congestion on this pathway and conducted experiments to discern the potential causes. All experimental data is exhibited in Table I, with all measurements taken on the receiver side. The sender and receiver are fixed on two logical cores of the same physical core.

Based on our understanding of the microarchitecture, we hypothesize that when there is intense contention for LFB/SQ resources due to a large number of active memory accesses and hardware prefetching, the transmission path bridging higher and lower-level caches may experience congestion.

TABLE I: Reverse-engineered results from analyzing causes for contentions in pathways from L2 to L1d and L3 to L2. The difference between Group A and Group B is that the receiver in Group A is allocated a buffer size of 128 KiB, while in Group B it is allocated 512 KiB.

No	L1d Miss	L2 Miss	L3 Miss	LFB Sat.	Time	Description
A1	$14.8*10^9$	$1.68*10^6$	$53*10^3$	$18.4*10^9$	$12.72s$	The sender and receiver access L2 cache at the same time.
A2	$12.4*10^9$	$0.08*10^6$	$32*10^3$	$9.8*10^9$	$4.73s$	The receiver accesses L2 cache alone.
A3	$14.3*10^9$	$0.26*10^6$	$52*10^3$	$4.1*10^9$	$7.20s$	The sender accesses L1 cache while the receiver accesses L2 cache.
A4	$12.4*10^9$	$0.31*10^6$	$230*10^3$	$0.85*10^9$	$3.90s$	With prefetchers off, the sender and receiver access L2 cache at the same time.

TABLE II: Reverse-engineered data analyzing the causes of congestion in the L2-to-L1i path.

No	L1i Miss	L2 Hit	Time	Cycles	Description
C1	$0.16*10^9$	$7.2*10^3$	$3.88s$	$17.07*10^9$	The sender and receiver fetch from L2 cache at the same time.
C2	$1.50*10^9$	$2.3*10^3$	$1.82s$	$7.99*10^9$	The receiver fetches from L2 cache alone.
C3	$0.44*10^9$	$14.7*10^3$	$3.36s$	$14.75*10^9$	The sender fetches from L1 cache while the receiver fetches from L2 cache.
C4	$0.14*10^9$	$10.2*10^3$	$4.27s$	$18.80*10^9$	With prefetchers off, the sender and receiver fetch from L2 cache at the same time.

Experimentation will be employed to confirm this supposition. In A1, a 128KiB buffer is allocated to both parties, allowing the sender and receiver to simultaneously access the L2 cache. In A2, only the receiver operates independently, also with a 128KiB buffer allocated for accessing the L2 cache. The code, as shown in Listing 1, uses larger stride lengths to mitigate the effects of L1 cache hits induced by prefetchers, aiming to maximize the number of memory access instructions that hit the L2 cache. At the same time, prefetching will also help us consume a portion of the bandwidth.

```
%assign number 0
%rep repetitions
    movzx eax, BYTE [rdx+STRIDE*number]
    movzx ebx, BYTE [rdx+STRIDE_2*number]
%assign number number+1
%endrep
```

Listing 1: Code for inducing L2-L1d and L3-L2 cache bandwidth congestion.

By comparing A1 and A2, we observed that the execution time of A1 is significantly higher than that of A2. However, this time increase could be attributed to multiple factors, including competition for front-end and back-end resources, a factor common to all hyper-threading timing side-channels, as well as L1d cache misses. It is necessary to rule out these factors.

In A3, the sender is allocated a 16KiB buffer, while the receiver retains a 128KiB buffer, allowing the sender to access the L1d cache and the receiver to access the L2 cache. Comparing A1 and A3, we found that A1's execution time remains higher than A3's, indicating that competition for front-end and back-end resources cannot sufficiently account for the difference in execution time. There are other factors that contribute to the high execution time of A1.

In A4, the configurations for the sender and receiver are

the same as in A1, but with the prefetchers disabled. We observe a significant decrease in execution time compared to A1. When comparing A1 and A4, we can identify two factors contributing to this reduction in time: fewer L1d misses and fewer instances of LFB saturation. However, when comparing A2 and A4, under the same number of L1d misses, A2 exhibits higher execution time even while having exclusive use of a whole physical core. This is due to the increased instances of LFB saturation in Group A2. Therefore, we conclude that prefetching and LFB saturation are the main causes of cache bandwidth congestion. This observation is more pronounced in Group B, where the impact of L1d misses appears to be even less significant. Due to space limitations, we will not provide a detailed description of the analysis process for Group B. The process is similar to that of Group A.

```
%assign number 0
%rep repetitions
    jmp .next_%+number
    nop11 nop11 nop11 nop11 nop11 nop7
    .next_%+number :;
    %assign number number+1
%endrep
.next_%+number :;
```

Listing 2: Code for creating bandwidth congestion from L2 to L1i. We have inserted long nop instructions to avoid backend execution bottlenecks caused by jump instructions.

Data from the L2 cache does not only feed the LFB and L1d caches, but also part of it enters the L1i cache. As a result, we analyzed whether the cache path from L2 to L1i could become congestion due to prefetching. Group C's configuration mirrors that of Groups A and B. By controlling the number of instructions to overwhelm the DSB, over 99.9% of all instructions in Group C decoded in MITE pipeline. The code is shown in Listing 2. This makes sure that, most of the time, the program is required to fetch instructions from caches other than the DSB. To avoid jump instructions from causing stalls in the backend, and therefore affecting time measurements, we inserted six long *nop* instructions, totaling 62 bytes, between each *jmp* instruction. Together with the *jmp* instruction, this forms a 64B code block. Results are shown in Table II.

In C1, both the sender and receiver access the L2 cache simultaneously. A comparative analysis with C2, wherein only the receiver accesses the L2 cache, reveals an elevated execution time in C1. This suggests the existence of contention. However, the source of this contention is not yet clear, leading us to conduct experiments in C3. In C3, the sender attempts to fetch instructions from the L1 cache as much as possible, while the receiver aims to fetch instructions from the L2 cache. The potential causes of time discrepancies could be L2 cache hits and prefetching. However, considering that an L2 cache hit confers a mere 20-cycle advantage, which stands in stark contrast to the final cycle count, we posit that prefetching, which culminates in cache bandwidth congestion, serves as the principal influencer of the time differentials. However, in the experiments conducted in C4, we did not observed the expected reduction in execution time after disabling the prefetcher. Despite ruling out the influence of caching factors, we posit that interference to the prefetcher occurs when simultaneously accessing the L2 cache, resulting in a decrease in bandwidth. This hypothesis is supported by a comparison of C1, C3, and C4. Similar observations were made during attacks on the cryptography libraries in Section V.

IV. Covert Channels

A. Threat Model

In our three covert channels, we presume that the sender and receiver are located on different logical cores within the same physical core. We suppose both parties are considered to belong to different security domains and lack legitimate communication channels, in line with existing covert channel research [3], [4], [10], [11]. Our covert channel protocol resembles traditional cache-based covert channels, but uniquely, it does not necessitate shared memory between the sender and receiver, nor does it require root privileges.

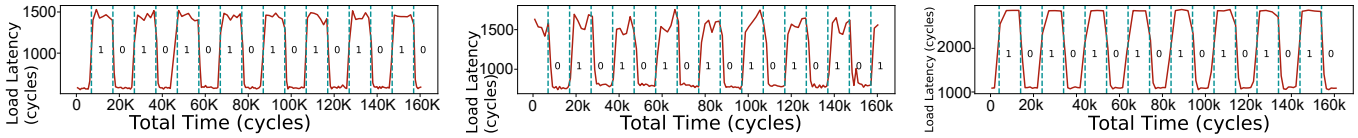
B. Experiments

The sender and receiver convey information through contention on the cache bandwidth. We encode a '0' as a state with no contention on cache transmission and a '1' as a state with contention. The receiver measures the time of memory access instructions, which consume a portion of the cache bandwidth. Consequently, when the sender transmits a '1', the memory access time for the receiver increases due to shared cache bandwidth usage. To synchronize both parties, we use a shared timestamp. There are other techniques [20] can replace this.

We first undertook a conceptual implementation of the covert channel to ensure accurate information transmission. In this scenario, the sender and receiver are threads running on two logical cores within the same physical core. In L2CC and L3CC, the sender and receiver engage in competition using the code from Listing 1, while both parties in LiCC utilize the code in Listing 2. The sender alternates between transmitting '1' and '0' every 10,000 cycles. In our three covert channels, '1' and '0' are distinguishable with clarity in Figure 1.

To assess the performance of our covert channels, we adopted a capacity metric, as referenced in [3], [20], to measure the trade-off between the bit error rate and transmission performance.

As depicted in the Figure 2, we present the performance of our three covert channels transmitting on i7-10700(Comet Lake, 4.00 GHz). The sender transmits 0 and 1 intermittently to the receiver. An elevation in transmission speed incurs a gradual rise in the bit error rate. We have recorded both the maximum capacity and the capacity when the bit error rate is less than 5% in Table III. Compared to related works [3], [10], our capacity ranks the highest among covert channels that do not depend on shared memory, achieving up to 10.37 Mbps and 10.02 Mbps. In comparison to channels that utilize shared memory, we are on the same scale as the top-performing channel, Streamline [14], without necessitating a large shared memory space of 64



(a) The transmission effect of L2CC. (b) The transmission effect of L3CC. (c) The transmission effect of LiCC.

Fig. 1: Transmission effects with a transmission interval of 10,000 cycles.

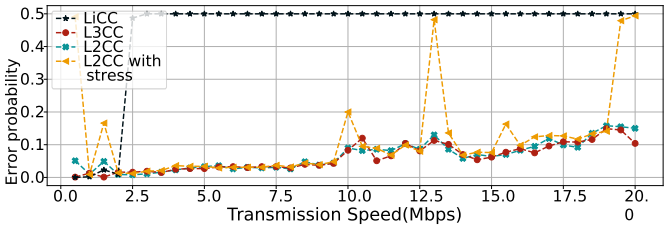


Fig. 2: The relationship between the transmission speed and the error probability for our three covert channels. We also measure the performance of L2CC in the presence of noise.

TABLE III: Comparison of covert channel capacities. Our channel capacities significantly exceed those of prior channels that did not require shared memory, and are on par with those channels that do require shared memory.

Name	Bandwidth	Error	Capacity	Anti-noise	Shared Memory
Lord of the Ring(s) [3]	< 6Mbps	<25%	4.14Mbps	Not tested	No
Frontend Bus [10]	0.7Mbps	0.81%	0.66Mbps	Not tested	No
	1.4Mbps	<10%	0.75Mbps	Not tested	No
StreamLine [14]	14.1Mbps	0.37%	13.57Mbps	Yes	Yes
Flush&FLush [21]	3.9Mbps	0.84%	3.60Mbps	Not tested	Yes
L2CC	18.0Mbps	9.2%	10.02Mbps	Yes	No
	9.5Mbps	4.4%	7.03Mbps	Yes	No
L3CC	20.0Mbps	10.4%	10.37Mbps	Yes	No
	9.5Mbps	4.3%	7.07Mbps	Yes	No
LiCC	2.0Mbps	1.0%	1.83Mbps	Not tested	No
L2CC with stress	15.0Mbps	7.6%	9.18Mbps	Yes	No
	9.0Mbps	3.9%	6.86Mbps	Yes	No

MB as Streamline does. LiCC's reduced speed results from its difficulty in distinguishing bit 0 from bit 1 after decreasing nops.

Please note that real-world noise may reduce the channel's transmission speed and elevate the bit error rate. During the transmission with L2CC, we secured "stress -m" on the same logical core as the sender. The results, as shown in the Figure 2, indicate that our covert channel is mostly unaffected, showcasing its strong resilience to interference and possible resistance to noise defenses [22].

V. Exploiting Cache Bandwidth Contention

A. Threat Model

We propose two types of attacks, where the attacker's code bears similarity to the receiver outlined in Section III. The threat model aligns with the covert channel section. We operate under the assumption that the administrator has set up the system to clear the victim's cache footprint during context switches in order to thwart preemptive scheduling attacks based on cache manipulation. The attacker consumes cache bandwidth via memory access instructions and measures their execution time. The attacker's buffer size is set to 128KiB. If the victim also executes memory access concurrently, the attacker's execution time will be extended. If the victim's memory access instructions are controlled by secrets, we can deduce the victim's secrets based on the execution time of the attacker. The experiments of Section V-B were conducted on the i5-8265U, and those in Section V-C on the i7-6700K. The experiments that combined with SpectreV2 required branch target injection. Although initial BTI attacks are mitigated in subsequent CPU generations, our work can be extended to newer CPUs via a fresh BTI attack vector [23]. Therefore, our side channel can be applied to all CPUs possessing an LFB structure.

B. Side channels

First, we demonstrate that our side channel can effectively leak information from cryptography libraries. As shown in Figure 3(a), many cryptography algorithms have adopted designs that follow such pseudocode, and there have been numerous side-channel attacks [1], [24] previously launched against such implementations in cryptography libraries. Both *Function_1()* and *Function_2()* are related to cryptographic operations, while the array holds the key.

In a naive implementation, we first assume this is the first execution of the victim's code. The victim needs to fetch the instructions and relevant data from the off-core cache to the on-core cache, thereby occupying part of the cache bandwidth and consequently affecting the execution time of the attacker. The victim initially retrieves instructions and data related to *Function_1()*, instigating the first contention with the attacker. When the key is 0, the victim bypasses the execution of *Function_2()*, hence avoiding any cache bandwidth contention. When the key is 1, the victim needs to execute *Function_2()*, thereby fetching the relevant data and instructions, which leads to the second contention with the attacker. In the second cycle of the loop, since *Function_1()* has already been executed, no contention is observed when *Function_1()* is executed again. Hence, the attacker can infer whether the victim's key is 0 or 1 based on the presence or absence of a second contention after the first.

As with many previous works [3], [25], the attacker can utilize techniques such as preemptive scheduling to expand the leak to multiple key bits. By interrupting the victim to induce a context switch, the victim's cache is cleared, returning the cache state to that before the first loop. Our implementation permits synchronization between the attacker and the victim. To achieve a cold start effect in experiments, the victim's cache is flushed before execution begins. We demonstrate the feasibility of using cache bandwidth contention for timing side-channel attacks on these two vulnerable versions of the RSA and EdDSA implementations. The two implementations are illustrated in Figure 3.

Figure 4a exhibits the result of the attack on the RSA in libgcrypt1.5.2 utilizing contention over L2 to L1 cache bandwidth. The attacker's timing function is the same as in Listing 1.

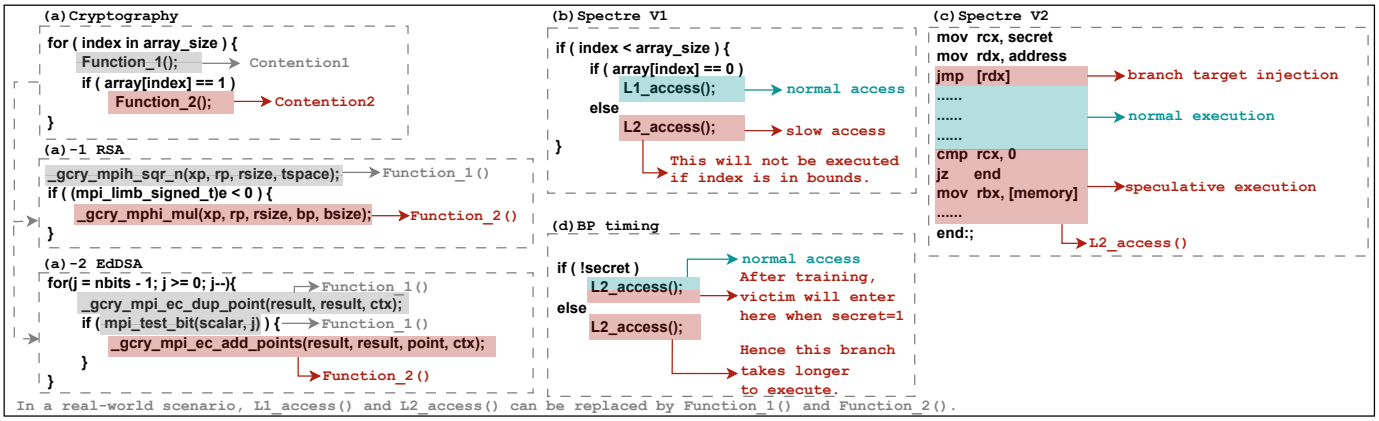
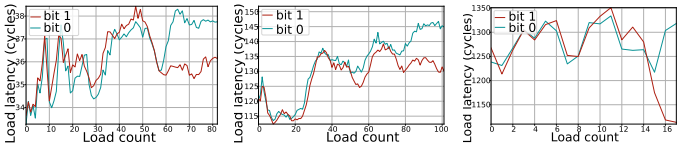


Fig. 3: Code patterns susceptible to exploitation through cache bandwidth contention.

The repetition is configured to 8 and the loop count to 1, balancing accuracy and speed considerations. Up to the 30th call, the timing is unstable, as we are still in the phase before entering the target gadget. From the 30th to the 60th calls, the contention between the attacker and the victim's *_gcry_mpih_sqr_n()* is documented. A discernible peak is observable due to the victim's unconditional execution of squaring. Intriguingly, after the 60th call, we witness a situation completely opposite to our expectations, which can be reproduced consistently. When the bit is 1 and executing *_gcry_mphi_mul()* requires fetching instructions and data into the on-core caches, the execution time is unexpectedly shorter than when no fetching occurs. We theorize that this may be because the attacker's prefetching remains undisturbed when *_gcry_mpih_sqr_n()* is executed twice consecutively (bit=0), whereas the prefetching is somewhat disrupted when *_gcry_mpih_sqr_n()* and *_gcry_mphi_mul()* are run in sequence, causing a reduction in cache bandwidth contention. As this counterintuitive peak can occur consistently, it does not hamper the leakage of secret information. For the RSA, we gathered 3000 traces, split them into training and testing sets, and trained a support vector machine. On the test set, the L2 cache bandwidth contention yields an accuracy rate exceeding 97.04%.



(a) Attack on the RSA algorithm leveraging L2-to-L1d bandwidth contention.

(b) Attack on the EdDSA algorithm using L2-to-L1d bandwidth contention.

(c) Attack on the EdDSA algorithm using L2-to-L1i bandwidth contention.

Fig. 4: Measured load latency during the victim's execution. The displayed values represent the total latency of multiple load instructions in the attacker's function.

We also targeted the EdDSA Curve25519 in libgcrypt 1.6.3. Due to space limitations, we will not provide a detailed description of the analysis process for this attack. For EdDSA, we adopted the same testing and training method as for RSA. Contention on the L2 cache bandwidth yielded an accuracy rate of 91.60% on the test set, while contention on the Li cache bandwidth resulted in a 75.9% accuracy rate. The lower accuracy rate could potentially be due to the lack of sufficient distinction in the absence of LFB and SQ structures.

C. Improvments to existing attacks

In this section, we improve SpectreV1, SpectreV2, and Branch predictor Timing attacks in an attempt to bypass existing defenses.

A side-channel attack comprises three steps: the attacker set up, the victim executes, and the attacker observes the corresponding data and decodes the secret from it. In SpectreV1, the attacker first trains the branch predictor by repeatedly executing the branch. The victim's execution leads to out-of-bounds memory accesses, which leaves a trace in the cache. The attacker then needs to observe the cache to obtain the secret. That is to say, without an observation method, even if the victim performs an out-of-bounds memory access, the attacker cannot obtain the secret. Four distinctive categories of defense are outlined in

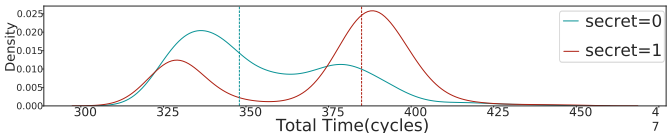


Fig. 5: Results of improvements to existing attacks. Distribution plot of L2CC combined with SpectreV2.

[17]: No setup, No Access without Authorization, No Use without Authorization, and No Send without Authorization. Except for defenses in the branch predictor, the remaining defenses aim to disrupt the covert channels in Spectre. Our improvements to Spectre attacks enable us to bypass these defense mechanisms except for the No Use without Authorization and Delay in No Send without Authorization, with a huge performance overhead.

We propose three new combinations to bypass the shielding of observation methods. The first one involves a modification to SpectreV1, which allows the attacker to avoid sharing memory with the victim. While the modifications to SpectreV1 typically necessitate specific gadgets within the victim's code, this constraint is eased when our covert channel is integrated with SpectreV2.

As illustrated in Figure 3(c), the victim initially moves the secret into the rcx register and jumping to the address stored in rdx. However, due to the influence of the attacker, the control flow is redirected to a gadget in the victim's code. The speculative execution causes cache access, affecting the attacker's

cache bandwidth. Therefore, by observing the execution time, the attacker can deduce the victim's secret. In comparison to the SMoTherSpectre [7], which utilizes port contention for transmission, our approach exhibits higher resilience against system noise and interference from other processes.

For branches that are dependent on secrets, a common code-level defense is branch balancing. This is demonstrated in Figure 3(d), which depicts a typical balanced branch structure. This defense strategy can be easily integrated into the cryptography libraries described in Section V-B. Nevertheless, we have managed to overcome this defense by training the branch predictor. When the secret is 0, the victim carries out regular L2 access. Yet, after the attacker trains the branch predictor, if the secret is 1, due to speculative execution, the victim enters the 'if' branch, competing with the attacker for cache bandwidth. The victim then re-executes the 'else' branch. As a result, the execution time for the attacker is longer than when the victim only executes the 'if' branch, thereby making it possible to distinguish the secret.

Due to space constraints, we have only presented the results of the integration with Spectre V2 in the Figure 5. After 1000 data collections, the secret becomes distinctly discernible. We reserve the exploration of the trade-off between data quantity and accuracy for future work.

VI. Conclusion

In this study, we analyze the root causes of cache congestion, establishing cache path congestion utilizing LFB/SQ contention and prefetch prediction failures. Such contention enabled us to construct three high-speed, noise-resistant covert channels. Our capacity ranks the highest among covert channels that do not depend on shared memory, yielding up to 10.37 Mbps and 10.02 Mbps. When measured against channels reliant on shared memory, our channels align closely with the performance of the highest-ranking channel, Streamline [14], without necessitating a large shared memory space of 64 MB as Streamline does. Additionally, we employed the contention to extract key bits from RSA and EdDSA implementations. Finally, we integrated our approach with existing Spectre and BP Timing attacks, circumventing some existing defenses. We have disclosed our results to Intel.

This work was supported by National Key R&D Program of China under Grant No. 2022YFB3103800 and National Natural Science Foundation of China under Grant No. 61972295.

References

[1] C. Percival, "Cache missing for fun and profit," 2005.

[2] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *2015 IEEE symposium on security and privacy*. IEEE, 2015, pp. 605–622.

[3] R. Paccagnella, L. Luo, and C. W. Fletcher, "Lord of the ring (s): Side channel attacks on the cpu on-chip ring interconnect are practical." in *USENIX Security Symposium*, 2021, pp. 645–662.

[4] S. Gast, J. Juffinger, M. Schwarzl, G. Saileshwar, A. Kogler, S. Franza, M. Köstl, and D. Gruss, "Squip: Exploiting the scheduler queue contention side channel," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2022, pp. 468–484.

[5] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown," *arXiv preprint arXiv:1801.01207*, 2018.

[6] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher *et al.*, "Spectre attacks: Exploiting speculative execution," *Communications of the ACM*, vol. 63, no. 7, pp. 93–101, 2020.

[7] A. Bhattacharyya, A. Sandulescu, M. Neugschwandtner, A. Sorniotti, B. Falsafi, M. Payer, and A. Kurmus, "Smotherspectre: exploiting speculative execution through port contention," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 785–800.

[8] M. Taram, A. Venkat, and D. Tullsen, "Context-sensitive fencing: Securing speculative execution via microcode customization," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 395–410.

[9] K. Loughlin, I. Neal, J. Ma, E. Tsai, O. Weisse, S. Narayanasamy, and B. Kasikci, "Dolma: Securing speculation with the principle of transient non-observability." in *USENIX Security Symposium*, 2021, pp. 1397–1414.

[10] K. Xu, M. Tang, H. Wang, and S. Guilley, "Reverse-engineering and exploiting the frontend bus of intel processor," *IEEE Transactions on Computers*, 2022.

[11] H. Wang, M. Tang *et al.*, "Cross-core and robust covert channel based on macro-op fusion," *Security and Communication Networks*, vol. 2023, 2023.

[12] J. Wan, Y. Bi, Z. Zhou, and Z. Li, "Meshup: Stateless cache side-channel attack on cpu mesh," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 1506–1524.

[13] I. Corporation, *Intel 64 and ia-32 architectures optimization reference manual*, 2023.

[14] G. Saileshwar, C. W. Fletcher, and M. Qureshi, "Streamline: a fast, flushless cache covert-channel attack by enabling asynchronous collusion," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 1077–1090.

[15] S. Van Schaik, A. Milburn, S. Österlund, P. Frigo, G. Maisuradze, K. Razavi, H. Bos, and C. Giuffrida, "Ridl: Rogue in-flight data load," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 88–105.

[16] S. Van Schaik, M. Minkin, A. Kwong, D. Genkin, and Y. Yarom, "Cacheout: Leaking data on intel cpus via cache evictions," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 339–354.

[17] G. Hu, Z. He, and R. B. Lee, "Sok: Hardware defenses against speculative execution attacks," in *2021 International Symposium on Secure and Private Execution Environment Design (SEED)*. IEEE, 2021, pp. 108–120.

[18] M. M. Thu, M. Méndez Real, M. Pelcat, and P. Besnier, " You Only Get One-Shot: Eavesdropping Input Images to Neural Network by Spying SoC-FPGA Internal Bus," in Proceedings of the 18th International Conference on Availability, Reliability and Security, Benevento Italy: ACM, 2023, pp. 1–7.

[19] J. Sepúlveda, M. Gross, A. Zankl, and G. Sigl, "Beyond Cache Attacks: Exploiting the Bus-based Communication Structure for Powerful On-Chip Microarchitectural Attacks," ACM Trans. Embed. Comput. Syst., vol. 20, no. 2, pp. 1–23.

[20] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, "Drama: Exploiting dram addressing for cross-cpu attacks." in *USENIX Security Symposium*, 2016, pp. 565–581.

[21] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+ flush: a fast and stealthy cache attack," in *Detection of Intrusions and Malware, and Vulnerability Assessment: 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016, Proceedings 13*. Springer, 2016, pp. 279–299.

[22] M. A. Mukhtar, M. Mushtaq, M. K. Bhatti, V. Lapotre, and G. Gogniat, "Flush+ prefetch: A countermeasure against access-driven cache-based side-channel attacks," Journal of Systems Architecture, vol. 104, p. 101698, 2020.

[23] E. Barberis, P. Frigo, M. Muench, H. Bos, and C. Giuffrida, "Branch history injection: On the effectiveness of hardware mitigations against $cross-privilege$ spectre-v2 attacks," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 971–988.

[24] B. Gras, C. Giuffrida, M. Kurth, H. Bos, and K. Razavi, "Absynthe: Automatic blackbox side-channel synthesis on commodity microarchitectures." in *NDSS*, 2020.

[25] D. Evtyushkin, R. Riley, N. C. Abu-Ghazaleh, ECE, and D. Ponomarev, "Branchscope: A new side-channel attack on directional branch predictor," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 693–707, 2018.