

Memory-aware Denial-of-Service Attacks on Shared Cache in Multicore Real-Time Systems

Michael Bechtel
mbechtel@ku.edu
University of Kansas
Lawrence, Kansas, USA

Heechul Yun
heechul.yun@ku.edu
University of Kansas
Lawrence, Kansas, USA

KEYWORDS

Denial-of-Service Attack, Shared Cache, Multicore, HugePage, Memory Address Mapping

ACM Reference Format:

Michael Bechtel and Heechul Yun. 2020. Memory-aware Denial-of-Service Attacks on Shared Cache in Multicore Real-Time Systems. In *Proceedings of ACM Conference (Conference'17)*. ACM, New York, NY, USA, 2 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

1 INTRODUCTION

In this paper, we propose *memory-aware cache DoS attacks* that can induce more effective cache blocking by taking advantage of information of the underlying memory hardware. Like prior cache DoS attacks, our new attacks also generate lots of cache misses to exhaust cache internal shared hardware resources. The difference is that we carefully control those cache misses to target the same DRAM bank to induce bank conflicts. Note that accesses to different DRAM banks can occur in parallel, and are thus faster. However, accesses to the same bank are serialized, and thus slower [5] and as each memory access request takes longer to finish, it would prolong the time it takes for the cache to become unblocked. We further extend these attacks to exploit HugePage support in Linux in order to directly control physical address bits and to avoid TLB contention, while mounting the attacks from the userspace.

1.1 Cache DoS Attacks

Recent works have demonstrated that the internal hardware structures of a non-blocking cache can be exploited to mount denial-of-service (DoS) attacks [3, 4]. Cache DoS attacks are software attacks that target cache internal hardware structures of a shared non-blocking cache. Generally, they are designed to generate as many cache misses and/or write-backs as fast as possible to overflow the cache internal hardware structures to induce cache blocking [3, 4]. When cache blocking occurs on a shared cache in a multicore processor, it affects all cores and can be devastating. For example, it has been shown that on a popular embedded multicore platform, the Raspberry Pi 3, a cache DoS attack could cause over 300X slowdown to a victim task [3].

Unpublished working draft. Not for distribution.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted by ACM, Inc., provided that the copies are not made for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Conference'17, July 2017, Washington, DC, USA

© 2020 Association for Computing Machinery.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM. \$15.00

<https://doi.org/10.1145/nnnnnnn.nnnnnnn>

2020-02-27 01:00. Page 1 of 1–2.

1.2 DRAM Bank-Aware Parallel Linked-List Attack

Due to the sequential access patterns of previous cache DoS attacks, they can be efficiently processed in memory which can result in reduced cache blocking durations and, by proxy, less impact to performance. To overcome these limitations, we propose a memory-aware cache DoS attack that performs random accesses over multiple linked lists while controlling each list's entries so that they are all allocated in the same DRAM bank. The rational is that when multiple accesses target the same bank, they will take longer to be serviced at the DRAM because of increased DRAM bank conflicts and frequent row switching.

```
1 int paddr_to_color(unsigned long mask, unsigned long paddr)
2 {
3     int color = 0;
4     int idx = 0;
5     int c;
6     for_each_set_bit(c, &mask, sizeof(unsigned long) * 8) {
7         if ((paddr >> (c)) & 0x1)
8             color |= (1 << idx);
9         idx++;
10    }
11    return color;
12 }
```

Figure 1: Physical address coloring used for creating enhanced cache DoS attacks.

To create such linked lists, creates a user-defined number of linked lists and populates them with addresses that map to the same DRAM bank. Figure 1 shows the code snippet that is used for finding such addresses. Specifically, it checks the value of each bit in a given address that corresponds to the set of bits specified in the *mask* bitmask, which is the platform's physical address bits that are mapped to DRAM banks. If the returned value is zero we add the address to the linked list as a new entry, otherwise we discard the address and continue. Since 2MB pages are used for memory allocation, the attack code can control both the physical and virtual addresses of the list entries. As a result, all of the linked lists generated, and their entries, will be allocated to the same memory bank and generate bank contention.

One notable shortcoming of the proposed memory-aware attacks is that they do not support in-order processing cores because they cannot concurrently traverse multiple linked lists. We instead target out-of-order core architectures that are inherently capable of generating multiple memory requests so that our attacks can still successfully and effectively generate cache blocking.

2 EVALUATION

In this section, we evaluate the effectiveness of the proposed memory-aware cache DoS attacks on two embedded multicore-based platforms using both synthetic applications.

2.1 Embedded Multicore Platforms

We deploy our DoS attacks on two embedded multicore platforms:

Platform	Odroid XU4		Raspberry Pi 4 Model B
SoC	Exynos5422		BCM2711
CPU	4x Cortex-A7 in-order 1.4GHz	4x Cortex-A15 out-of-order 2.0GHz	4x Cortex-A72 out-of-order 1.5GHz
Private Cache	32/32KB	32/32KB	32K/32K
Shared Cache	512KB (16-way)	2MB (16-way)	512KB (16-way)
Local MLP	1	6	6
Global MLP	4	11	19
Memory (Peak BW)	2GB LPDDR3 (14.9GB/s)		4GB LPDDR4 (25.6 GB/s)
DRAM bank bits (Bitmask)	13, 14, 15, 16 (0x1E000)		8, 11, 12, 13, 14 (0x7900)

Table 1: Compared embedded multicore platforms.

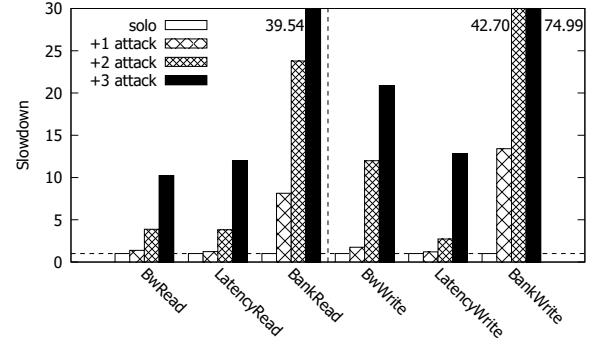
an Odroid XU4 and a Raspberry Pi 4 Model B. The Odroid XU4 employs a big.LITTLE processor configuration comprised of a smaller 4xCortex-A7 [2] in-order core cluster and a larger 4xCortex-A15 [1] out-of-order core cluster. Note that we do not use the Cortex-A7 cluster on the Odroid XU4 as we are primarily focused on out-of-order architecture designs due to their inherent ability to generate increased traffic at both the shared LLC and memory levels. The second platform we test, the Raspberry Pi 4, only equips a single cluster of 4x Cortex-A72 out-of-order cores. All platform specifications can be seen in Table 1.

2.2 Synthetic Workload

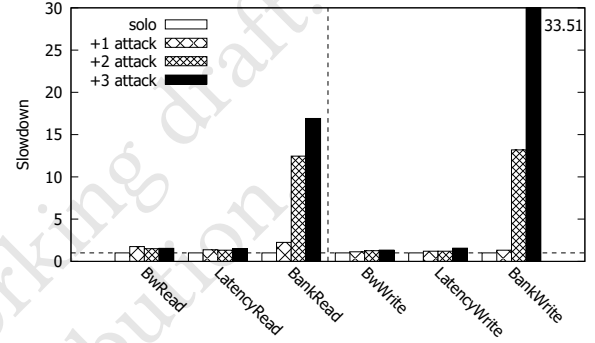
The experimental setup is as follows: we run each victim task alone on a single core, Core 0, to measure its solo response time. We then run the victim task alongside up to three instances of each attacker, scheduled on Cores 1-3, and measure the response times to determine the slowdown each attack caused on the victim relative to the solo case. For the victim task, we use the *latency-mlp* synthetic workload from the IsolBench suite [4]. Note that we configure the victim to fit inside the LLC of each tested platform.

For the attackers, we employ three cache DoS attack types: (1) Bw, a sequential access attack we’ve used in previous works [3, 4], (2) Latency, a random access attack with no constraints on its linked list entries, and (3) Bank, our proposed attacks described in Section 1.2. Each attack can then be configured to perform reads or writes, so we deploy a total of six attacking tasks. For all attacking processes, we configure their working set sizes to fit inside of DRAM such that they can effectively generate shared cache misses.

Figure 2 shows the impacts of cache DoS attacks to the *latency-mlp* victim. As expected, the Bank attacks have noticeably more impact compared to the older DoS attacks. On the Odroid-XU4, BankRead and BankWrite attackers achieve up to ~40X and ~75X slowdowns, respectively, to the *latency-mlp* victim task’s performance. Likewise, they cause ~17X and ~34X slowdown, respectively, when run on the Pi 4.



(a) Odroid XU4(A15)



(b) Raspberry Pi 4 (A72)

Figure 2: Impacts of cache DoS attacks to the *LatencyRead* victim on two multicore platforms. The victim (*LatencyRead*) runs on Core 0 while the attackers (X-axis) run on Core 1-3.

3 CONCLUSION

In this paper, we introduced memory-aware cache DoS attacks that leverage a system’s memory address mapping information and HugePage support to induce prolonged cache blocking by intentionally creating DRAM bank congestion. From experimental results on two popular embedded multicore platforms, we show that our new DoS attacks can generate significantly higher timing impact to cross-core victim tasks compared to prior cache DoS attacks. For future work, we plan to launch our attacks on platforms that employ more sophisticated XOR address mapping schemes and evaluate their feasibility on server and cloud-based platforms.

REFERENCES

- [1] ARM. *Cortex™-A15 Technical Reference Manual, Rev: r4p0*, 2011.
- [2] ARM. *Cortex™-A7 Technical Reference Manual, Rev: r0p5*, 2012.
- [3] Michael G Bechtel and Heechul Yun. Denial-of-service attacks on shared cache in multicore: Analysis and prevention. In *IEEE International Conference on Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2019.
- [4] Prathap Kumar Valsan, Heechul Yun, and Farzad Farshchi. Taming Non-blocking Caches to Improve Isolation in Multicore Real-Time Systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2016.
- [5] Heechul Yun, R. Pellizzoni, and P. Valsan. Parallelism-Aware Memory Interference Delay Analysis for COTS Multicore Systems. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 184–195, 2015.