Defeating the Recent AnC Attack by Simply Hashing the Cache Indexes — Implemented in a BOOM SoC

Wei Song, Rui Hou, Dan Meng
Institute of Information Engineering, Chinese Academy of Sciences
89A Minzhuang Road, Haidian District, Beijing, P.R China 100195
Email: {songwei, hourui, mengdan}@iie.ac.cn

I. The AnC Attack

A recently proposed attack, namely the ASLR⊕Cache or simply AnC [1], provides a universal method to bypass the ASLR in all major browsers by utilizing existing side-channels on memory management units (MMU) and caches. It is claimed that the AnC attack exploits the fundamental properties of cache based computer architectures; therefore, the side-channels are difficult to eliminate without significant performance degradation. As a result, ASLR is fundamentally insecure.

The AnC attack describes a principled way to bypass the address space layout randomization (ASLR) defense in all major browsers. The key to successfully bypassing ASLR is to first obtain the virtual address of a user page. In AnC, this goal is equivalent to inferring the VPN of an attacker created variable inside this user page. If this user page is recently accessed by the attacker, there is a good chance that the four PTEs of this user page is cached in the L1 data cache. Since VPN is used in the PTW procedure as the set of PT offsets and these PT offsets are also partially used as the cache indexes denoting the cache sets holding the four related PTEs, the attacker can launch contention-based attacks to identify these cache sets which expose the PT offsets and ultimately the VPN. To be specific, the attacker needs to acquire three pieces of information to infer the whole VPN: the cache indexes of the four related PTEs, the offset of each PTE inside its cache line, and the mapping between PT offsets and PT levels. The first two pieces of information reveal the PT offset and the third piece of information tells the order in assembling the four PT offsets into the VPN. Let us consider an AnC attack on the L1 VIPT cache. To locate the four related PTEs and learn their cache indexes, the attacker scans all cache sets using the Evict+Time attack. Each cache set is first evicted and then tested through an intentionally triggered PTW by flushing TLB and accessing the target variable. If the evicted cache set contains a related PTE, PTW would miss causing a long delay. The rest two pieces of information are acquired using a sliding technique [1] which combines multiple rounds of aforementioned scan of cache sets. In each round, the attacker locates the four related PTEs of a variable with a known distance away from the target variable. According to the relationships (and their patterns) between the distance and the changes on the cache indexes of the four related PTEs, the attacker is able to collect enough information to figure out the cache line offsets, and the mapping between PT offsets and PT levels.

In summary, the AnC attack relies on several assumptions on architecture and micro-architecture features:

- A direct mapping between VPN and the PT offsets in the page tables.
- Uniformed caches that stores both data and page tables.
- A direct mapping between addresses and cache indexes.

II. Defending the AnC attack

According to the 3 assumptions required by AnC, we believe the key in defense is to break the 3rd assumption: the direct mapping between memory addresses and cache indexes.

A straightforward way to to break this direct mapping is to apply a hash function on the cache index calculation. This hash function should map each $A[s+5:6]$ unambiguously to a CI with a low hardware overhead. The simplest way to do it, is to hash the address with a key unaccessible to the attacker.

Considering most attacker run in a virtual space without direct access to the page table, the physical page number or the higher portion of the physical address is already a secrete. Therefore, a cache can break the direct mapping by simply hashing addresses with the higher digits from themselves:

$$CI = A[2s+5:s+6] \oplus A[s+5:6] \tag{1}$$

For a normal VIPT L1 cache, Equation 1 can be described using the virtual address (VA) and physical page number (PPN) as well:

$$CI = PPN[5:0] \oplus VA[11:6] \tag{2}$$

Here we assume the usual configuration of making the set number equal with the number of cache lines in a page ($2^s \cdot 64B = 4KB$).

This defense is most effective with an OS constantly randomizing the physical to virtual page mapping. The hash function uses PPN as the hash key. However, the PPN for adjacent virtual pages might be related as memory allocators tend to allocat consecutive physical pages. This might give attackers a means to decipher the PPN and disarm the defense.

The support of large pages is another feature which can weaken RCL-Hash. On a typical 64-bit machine, a large page is usually $2MB$. This provides an attacker with a guaranteed virtually and physically consecutive space capable of inferring

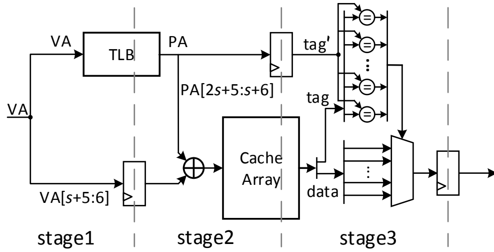


Fig. 1. Implementation of RCL-Hash.

the hash keys for an 8-way cache as large as 2048 sets (assuming the FIFO replacement policy).

In summary, the effectiveness of the defense depends on how much an OS tends to allocate physically consecutive pages. If a system deliberately randomizes the physical to virtual page mapping and disables the large page support, the simple hash is effective in defending the AnC attack.

III. Implementation

The proposed RCL schemes have been implemented on the opensourced BOOM system-on-chip (SoC) [3], [4].

The hashed cache index implemented in the L1 VIPT cache is depicted in Fig. 1. To implement the defense, the PA is required to be translated before accessing the cache array. The virtual to physical address translation is made to operate one cycle before accessing the cache array. Although this serialization increases the cache latency by one cycle, its impact on performance should be limited as the out-of-order and speculative execution in a superscalar processor is able to partially hide the cache access latency.

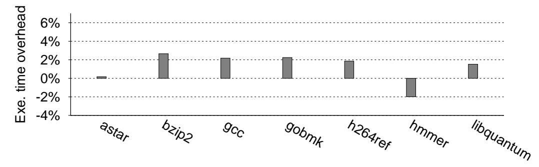
IV. Evaluation

A subset of the SPEC 2006 benchmark cases run successfully on the BOOM SoC. Fig. 2a reveals prolonged execution time compared with the original BOOM SoC. On average, the execution time increases by 1.0%. The execution time is prolonged because the cache access latency of the L1 caches is increased due to the serialization of the virtual to physical address translation and the access to cache array. For certain cache configurations, the regular access pattern of an application might cause extra conflict misses. A remapped cache layout might provide a more balanced data distribution which reduces conflict misses, resulting in a slightly reduced execution time.

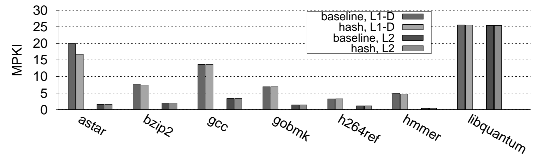
To extract the actual cache miss rates from the hardware, hardware performance counters are added to all levels of caches. These counters constantly record the numbers of cache accesses, cache misses and write-backs by monitoring the cache control logic. Fig. 2b reveals the miss per kilo instructions (MPKI) collected from the L1 data cache and the L2 cache. There is no significant change in the MPKI on both cache levels.

V. Conclusions

A new defense is posposed to defend against the recent AnC attack. It remaps the cache layout using a hash function



(a) Execution time overhead compared with the baseline.



(b) Cache miss per kilo instructions (MPKI).

Fig. 2. Running SPEC 2006 on FPGA.

on the cache indexes. This is an effective defense when the OS actively randomizes its physical to virtual page mapping. Implemented in the BOOM SoC, the results show that the defense incurs a small overhead in execution time.

Acknowledgement

The authors would like to thank Christopher Celio (the designer of BOOM) for his generous help on running the SPEC 2006 benchmark on the BOOM SoC.

References

[1] B. Gras, K. Razavi, E. Bosman, H. Bos, and C. Giuffrida, "ASLR on the line: Practical cache attacks on the MMU," in *Proc. of the Network and Distributed System Security Symposium*, 2017, p. 15.

[2] K. Z. Snow, F. Monrose, L. Davi, A. Dmitrienko, C. Liebchen, and A.-R. Sadeghi, "Just-In-Time code reuse: On the effectiveness of fine-grained address space layout randomization," in *Proc. of the Symposium on Security and Privacy*, 2013, pp. 574–588.

[3] C. Celio, D. A. Patterson, and K. Asanović, "The Berkeley out-of-order machine (BOOM): An industry-competitive, synthesizable, parameterized RISC-V processor," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-167, 2015, http://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-167.html.

[4] K. Asanović, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz, S. Karandikar, B. Keller, D. Kim, J. Koenig, Y. Lee, E. Love, M. Maas, A. Magyar, H. Mao, M. Moreto, A. Ou, D. A. Patterson, B. Richards, C. Schmidt, S. Twigg, H. Vo, and A. Waterman, "The Rocket chip generator," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17, 2016, http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html.