POSTER: Trapping Spectres in Speculation Domains

Isaac Richter
Department of Electrical and Computer Engineering
University of Rochester
Rochester, NY, USA
isaac.richter@rochester.edu

Yufei Du
Department of Computer Science
University of Rochester
Rochester, NY, USA
ydu14@ur.rochester.edu

John Criswell
Department of Computer Science
University of Rochester
Rochester, NY, USA
criswell@cs.rochester.edu

***Abstract*—This work introduces Speculation Domains, a method for preventing speculation from being used in Spectre and Meltdown side-channel attacks.**

I. Introduction

Modern processors speculatively execute instructions to achieve high throughput. Instructions executed in error are squashed, rolling back their effects. Speculation side-channel attacks, such as Spectre [1] and Meltdown [2], leverage speculative execution to cause the processor to execute instructions that would not have been executed to load secret data into processor registers. These attacks then leak this data to the attacker via cache side channels.

Speculation-based attacks are dangerous because they leak sensitive data at high bandwidth [2] and because disabling speculation incurs a substantial performance penalty. This work attempts to prevent speculation from being used to attack and exfiltrate data via cache side channels utilizing either Flush+Reload [3] or Prime+Probe [4].

II. Speculation Domains

We propose *speculation domains*, a design which guarantees that cache lines loaded in a given domain are not accessible to code running in other domains until the instruction that caused them to be loaded is retired without exception. If multiple domains attempt to access the same cache line, the processor loads the line multiple times.

Our design prevents speculative side-channel attacks [1], [2] from leakage data via the cache by ensuring that the speculatively-loaded line does not appear as having been loaded by other domains. Nonetheless, by allowing loaded lines to be accessible by all speculation domains after the load is no longer speculative, we avoid the need for long-term data duplication.

Our design adds speculation domains as a new instruction set feature. All requests, both data and instruction, are tagged with the current speculation domain of the thread making the request. This is done for both memory requests and instruction fetches. When any instruction commits, all cache lines it used are promoted to the global domain. If all instructions that used a cache line are squashed, a request is sent to the cache to evict that line.

Cache tags are extended to include the speculation domain associated with the line. When processing requests, the cache must either match the request's domain to that of the line, or the line must be in the global domain. On a miss, the speculation domain is passed along to the lower-level cache (closer to the memory) or memory controller. The memory system does not coalesce requests from different domains. This can result in multiple outstanding requests for the same physical memory. This is desired behavior: the latency of loading the data from the lower-level cache is the same regardless of whether code in another domain sent a request for the same cache line.

When a line within a cache is promoted to the global domain, any non-global instances thereof in that cache are removed. Because writes always go to the cache upon commit, dirty lines are always already in the global domain, so this does not cause consistency issues. Promotions to the global domain are transmitted down through the hierarchy (toward the memory controller). Evictions due to squashed instructions are also propagated, but only if no sibling cache also holds the line under the same domain (the lower level cache tracks how many of its immediate higher-level caches have copies of the line, so it knows when it can evict its copy).

To handle Prime+Probe attacks [4], where the attacker first fills the cache and then checks for evicted lines, we reserve a logical portion of cache ways for speculatively-loaded data (similar to dynamic partitioning, for example in SecDCP [5]). The lines are arranged such that each domain will have dedicated ways, preventing any collisions between domains (and leakage of speculated loads to other speculation domains). As with the base case, lines are still marked as non-speculative once the related instruction is committed. If the destination set is otherwise full, the existing cache eviction mechanism will choose a line to evict, and re-purpose as a speculative line.

III. Conclusion

We believe that speculation domains will mitigate speculation-based attacks while still enabling high performance. We also believe that speculation domains can be used to mitigate side channels in other processor components e.g., TLBs. How software can best use speculation domains is an open question about which we hope to spark discussion.

REFERENCES

[1] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," 2018.

[2] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yaromn, and M. Hamburg, "Meltdown," 2018.

[3] Y. Yarom and K. Falkner, "FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack," in *Proceedings of the 23rd USENIX Security Symposium*, ser. SEC'14. Berkeley, CA, USA: USENIX Association, 2014, pp. 719–732. [Online]. Available: http://dl.acm.org/citation.cfm?id=2671225.2671271

[4] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: The case of AES," in *Proceedings of the 2006 The Cryptographers' Track at the RSA Conference on Topics in Cryptology*, ser. CT-RSA'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 1–20.

[5] Y. Wang, A. Ferraiuolo, D. Zhang, A. C. Myers, and G. E. Suh, "SecDCP: Secure dynamic cache partitioning for efficient timing channel protection," in *Proceedings of the 53rd Annual Design Automation Conference*, ser. DAC '16. New York, NY, USA: ACM, 2016, pp. 74:1–74:6. [Online]. Available: http://doi.acm.org/10.1145/2897937.2898086