CMP SCI 635: Modern Computer Architecture

Secure Hierarchy-Aware Cache Replacement Policy (SHARP): Defending Against Cache-Based Side Channel Attacks

Name: Kunjal Panchal Date: 12th Nov, 2019 Student ID: 32126469

Paper: 18 – Security [Yan17]

Strengths:

1. Region-based cache partition techniques have a relatively smaller performance overhead than process-based techniques, since they try to maximize the number of dynamic accesses to the shared cache while maintaining sufficient isolation. This can prevent spying without changing Cache replacement policy.

2. The pathological case when many referenced lines across all cores map to the same set in both private and shared caches, and the shared-cache associativity is not enough. While possible, this case is rare, only temporary, and only affects the relevant cache set. So, the only loophole of SHARP is very less likely to occur.

Weaknesses:

1. SHARP's idea for protecting against cache-based attacks might degrade performance of a genuine neighbour process by not giving up on enough cache lines if it finds that most of the "victim's" data is in private cache too. The scheme sounds unfair towards a neighbour who is suspected of spying (but it might actually just want its fair share of memory).

2. In SHARP, if spy tries to evict another core's cache line, and ends up evicting its own; the spy will still know that a cache line which it was targeting by its latency of load.

Questions/Assertions:

1. How does an attacker determine what memory address to probe? In case of Meltdown, we flush N locations and reload one location in the attack that corresponds to the value of kernel memory we just read. Then we probe all N locations to find which location was reloaded. Does all evict+reload attacks work the same way?

2. We have this recent upsurge of side channel attacks, but I reckon, all of them exploit RSA, old EC or AES-with-lookups. I wonder if there ever will be such attacks on modern crypto that conforms to the rule of no secret-based indexing or branching.

3. A variant of all this can be a "*POWER*" attack. Suppose all I can do is trace the current usage of a CPU as victim works. Now I mess with the cache and see how the current usage changes over time relative to a baseline [e.g. victim doing the operation without attacker messing about]. Now I can see that X cycles into the computation you used Y% more power, so chances are something I did to the cache there affected the work at that precise time. Since I know the algorithm just not the secret variables, I can guess what the secrets were.

4. Intel has added CLWB. CLWB does a writeback flush that doesn't invalidate (unlike CLFLUSH), and serializes to SFENCE in addition to MFENCE giving it "release" semantics. While this is useful for controlling visibility with respect to persistent memory, it's also just an optimization over CLFLUSH, albeit a significant one.

5. What happens after OS is notified that some potential spy has passed local core threshold for generating inclusive victims? Does OS kill the process?

6. Why can't we just restrict FLUSH instructions to kernel processes only? Wouldn't that stop spy processes from doing the side channel attack on the L3 cache?