Paper: LLC Side-channel attacks are practical.

→ Abstract:

- presents effective implementation of Prime+Probe side-channel attack on LLC.
- demonstrates the attack on cross-core, cross-VM and measure the capacity of covert channel
- their technique is independent of OS weaknesses, VMM, shared memory in achieving high attack resolution

→ Introduction:

- working of IaaS and how system resources are shared b/w co-hosted VMs.
- LLC is good candidate to mount an attack as its shared b/w all the cores but due to large size and slow access, temporal resolution of observable events and thus channel bandwidth
- Adaptation of Prime+Probe possible to mount on LLC by exploiting:
  (a) inclusive caches (outside control)
  (b) large page mappings (controllable)
- Only other assumption: attacker and victim are co-hosted on the same processor.

- Contributions:
  - ↳ async. Prime+Probe that doesn't require sharing cores or memory b/w attacker and victim., doesn't exploit VMM weaknesses
  - ↳ two techniques for efficient attacks:
    - (i) an algo. to probe exactly one ~~set~~ cache set w/o knowing VA-mapping
    - (ii) use temporal access patterns instead of spatial access patterns to identify victim's security-critical accesses.
  - ↳ cross-VM covert timing channel achievable bandwidth ⇒ 1.2 Mbps

→ Background

(A) Virtual Address space and large pages

- Virtual environments have 2 levels of address space translation
  - (i) maps process's VA to guest's notion of PA (i.e. VM's emulated physical memory)
  - (ii) maps guest PA to PA of the processor.

(B) System model and cache architecture.

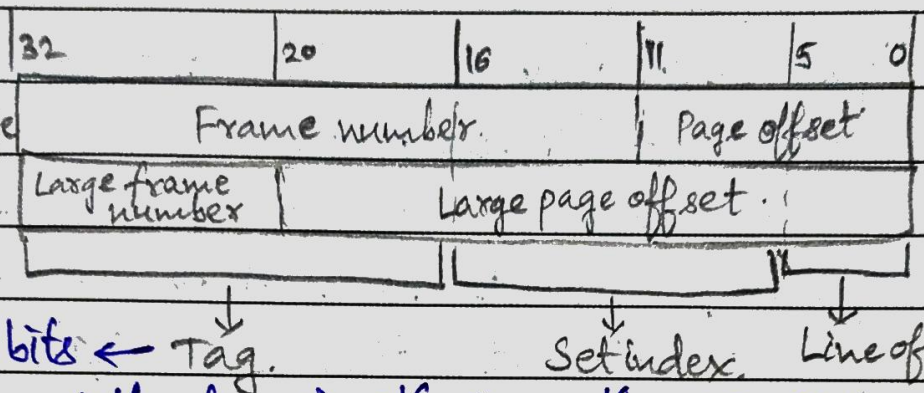
[1] Cache hierarchy

- Higher-level caches are closer to processor core, are smaller but faster than lower-level caches that are closer to main memory.
- Typically, 2 private top-level caches, one for data and instructions, called L1 caches.
- Typical L1 cache size = 32 KiB with 4-cycle access time.
- LLC is shared among all cores and is a unified cache, i.e. holds both data and instructions.
- LLC sizes ⇒ ~40 MB, access latency ~40 cycles
- L1 is typically indexed by virtual address, while other caches are indexed by physical address.

[2] Cache access

- Caches are organized in fixed-size lines
- Typical line size B = 64 bytes;
  $\log_2 B$ = line offset (lowest-order bits of address) are used to locate data in the cache line.
- W-way set-associative: S sets of W lines each
- set-index: $\log_2 S$ consecutive bits starting from $\log_2 B$ is used to locate cache set.

- remaining bits = tag for each cache line



correct ??

it means W (no. of ways) = 2^{16} ; since $2^{16} \times 2048 \times 64B = 8GiB$.
(sets) (line size)

Cache indexing for an 8 GiB address space.

A cache (or slice) contains 2048 (2^{11}) sets with 64B (2^6) lines

- after locating the cache set, the tag field of the address is matched against the tag of the W lines in the set to identify if one of the cache lines is a cache hit.

- Cache contention: since memory is much larger than the cache, more than W memory lines map to the same cache set → this means $W \neq 2^{16}$, no?? but lesser value

- Inclusiveness property: L_{i+1} cache holds a strict superset of the contents of the L_i.

→ Challenges in attacking the LLC

(A) Attack model

- Target info. leakage in virtual env. eg. IaaS clouds.
- Assumptions made:
  - ↳ attacker VM co-resides with victim VM on same multi-core processor
  - ↳ attacker knows some crypto s/w is running inside victim VM
- Assumptions not made:
  - ↳ any vulnerability in VMM
  - ↳ attacker and victim share a core, share memory or attacker synchronizes its execution with victim.

(B) PRIME + PROBE

- a general technique for an attacker to learn which cache set is accessed by the victim VM
- LLC-based cross-core cross-VM attack is based on above technique.

- Flow of PRIME + PROBE
  - ↳ attacker A runs spy program to monitor cache usage of victim V as follows:
    - → PRIME: A fills one or more cache sets with its own code/data
    - → IDLE: A waits for a pre-configured time interval while V executes and ~~utt~~ utilizes the cache.
    - → PROBE: A continues execution and measures the time to load each set of his data/code that he primed. If V has accessed some cache sets, it will have evicted some of A's lines which A observes as increased memory access latency for those lines.

didn't understand :"

↳ As the PROBE phase accesses the cache, it doubles as a PRIME phase for subsequent observations.

(C) Overview of challenges for efficient PRIME+PROBE attack on LLC

↳ (i) Visibility into one core's memory accesses from another core via LLC
[D. Visibility of processor-memory activity at LLC]

- L1 and L2 will satisfy most of the processor's memory accesses ⇒ LLC has less visibility into victim's memory activity than the L1.
- If manipulation in LLC by attacker does not impact state of higher-level caches used by the victim VM, the victim's accesses to its code/data will never reach the LLC and hence hidden to attacker.
- Overcome: leverage (take max. advantage) of cache inclusiveness, that replaces victim data from complete cache hierarchy w/o access to any victim's local caches.

↳ (ii) Significantly longer time to probe large LLC
[E. Infeasibility of priming and probing whole LLC]

- For L1, entire cache is primed and probed and then ML algo. is used to identify spatial locality associated with victim's memory activity.
- Since LLC is very large than L1, the above is infeasible for LLC.
- Overcome: first determine very few cache sets corresponding to victim's accesses and then monitor only those cache sets during prime/probe step, instead of monitoring whole LLC.

[F]

↳ (iii) Identifying cache sets relevant to victim's code and data.

- attacker doesn't know the virtual addresses in victim's address space and has no control on how these VAs are mapped to PAs.
- Overcome: scan entire LLC monitoring one cache set at a time, look for temporal access patterns that are consistent with victim's accesses.

- these specific temporal access patterns depend on the algorithm used.

(iv) constructing eviction set that occupies exactly one cache set in LLC, w/o knowing the address mappings.

[G. Eviction set to occupy exactly one cache set]

- to monitor victim's accesses to one specific cache set, attacker needs to be able to occupy that specific cache set
- to achieve this, attacker can construct an eviction set containing a collection of memory lines in its own address space that all map to a specific cache set.
- the eviction set must contain W memory lines in order to evict one complete set, since a cache set contains W cache lines
- as LRU will replace older lines before loading other lines, touching each line once will evict any prior data in the set.