Last-Level Cache Side-Channel Attacks are Practical

Hyperthreads (**6**)
Sumon Nath
(21Q050007)

I. Summary

In this paper, the authors propose an implementation of prime+probe attack against the last level cache, on a cross-core, cross-VM setup. They implement this attack on single cache set instead of the entire LLC to reduce probe resolution. They also measure the covert channel capacity to study the probe resolution. Finally they show the practically of the attack by demonstrating them on two versions of GnuPG to leak the secret key.

II. Details

- The attack model assumes that the victim uses some cryptographic library and the attacker knows this crypto software. It also assumes that the victim and the attacker VM runs on different cores of the same system that shares the LLC.
- Instead of priming/probing the entire cache, they do it on a particular cache set to reduce the attack resolution time.
- They use an algorithm to find a collection of eviction sets across all slices to counter the problem of LLC slicing. They also use pointer-chasing technique to nullify the effect of prefetching.

III. Positive

- The covert channel bandwidth goes as high as 1.2 Mb/s with 22% error rate and achieves a bandwidth of 600 Kb/s with error rate of 1 %.
- Demonstrates the practicality of the attack on two versions of GnuPG to leak the secret key.

IV. Negatives

- Authors assumes the LLC to be inclusive. In case of other type of caches like exclusive and non-inclusive this attack will not be practical as victim will get hits in the higher cache levels.
- This is a minor point. Large pages is one of the necessary assumptions for the attack to work. The authors mentioned that large pages are suited to applications with large memory footprints, as large pages use TLBs efficiently. A plot confirming the same would be helpful.
- They counter the problem of LLC slicing keeping in mind only Intel's hash functions but does take into account other LLC slicing techniques.
- To avoid attacker's self-eviction while probing they reverse the traversal of eviction set assuming the LRU replacement policy is used. In case an advanced replacement policy is used like DRRIP this technique will not work.

I

Last-level Cache Side-Channel Attacks are Practical

Interim Notes

~~1st / Pass~~

Key-phrases (abstract) : – measure capacity of covert channel
– cross-core, cross-VM attack
– high attack resolution

Intro :

Background – IaaS ~~[illegible]~~
└ multiple VMs – high resource utilization by sharing resources.

Problem – Shared physical resources, ~~may~~ leads to interference between co-hosted VMs. making them vulnarable to attaks.

- ~~LLC~~ Proposes an LLC based attack instead of L1D, as it is private to each core, & VMs ~~are~~ ~~generally~~ usually run on diff. cores.
- ~~[illegible]~~ ~~[illegible]~~ Problem with LLC based attacks – low leakage channel bandwidth, as LLC is slow ~~& temporal~~ ~~resoluti~~ which leads to low temporal resolution of observable events.

Assumptions : – inclusive caches (what about other types??)*
– Large pages (valid reason given in paper).

Contributions : methodology/ idea
- async. PRIME+PROBE on LLC. robust to unknown. unknown LLC hashing schemes.
- probes exactly one cache set. without knowledge of address mapping.

evidence/ experimental results.
- covert channel BW – 1.2 Mb/s
- attack showing key extraction from secret-dependent execution path, as well as secret-dependent data access patterns.

Imp points in Background:

1. Large pages are suited to applications with large-memory footprints, as large pages use TLBs efficiently.
   (W) (A plot confirming the above statement will be helpful).

* Background has a bit too much details.
  (Maybe included, as the paper is published in a security-related conference).

→ ~~How does~~
→ Brief idea about LLC slices & hashing.

Problems of prime+probe attack:

1. LLC has low visibility of victim's memory activity.
   ↳ sol: cache inclusiveness
   What about ~~other kinds~~ ~~&c~~ exclusive caches?

2. Infeasible to prime & probe entire LLC.
   ↳ sol: monitor only a few cache sets (security-critical).
   ↳ How to identify such sets?
   ↳ ~~ident~~ by identifying temporal access patterns ~~for~~ for each set which are application specific.
   Discussed in section VI & VII

3. challenges of creating eviction set:
   — LLC physically indexed. Targetting a specific set is hard as VMM's mapping is not accessible as well as sliced cache complicates the attack.

4. Probing LLC set is about one order of magnitude slower than probing L1.

Eviction set construction

main assumption — large pages (previously mentioned)

- To counter the problem of LLC slicing, they use an algo to find eviction sets.
  The algorithm creates a conflict set, which is a union of eviction sets for all the slices.

(*) Designed specifically keeping in mind Intel's hash function.
What about other LLC slicing techniques?

PRIME + PROBE:

- Uses pointer-chasing technique.
  ↳ this nullifies the effect of prefetching

Optimizations:

1. Avoiding attacker's self-eviction by ~~using~~ reversing traversal during probe ~~stage~~. (~~Need~~ caused by LRU policy)

(*) What about other repl. policies?
→ this approach will not work with modern repl. policies

Interaction with higher-level cache:

1. During probe, hits in L1/L2 ~~on~~ creates noise
Instead of measuring the total probe time, they measure probe time of every load.

(*) How will this reduce noise?
- Explanation is not satisfactory.

Channel capacity, error rate

→ ~~If~~ I feel the experiment does not model ~~the~~ a practical situation very well.

(*) Eg. The sender accesses a line1 continously for some time Tm ~~for~~ to indicate bit "1" then pauses for Time Tp then accesses another line 0 for Tm to indicate bit "0".
Such uniform accesses may not occur in real situation.

Although I'm not sure about this, this may be a standard way of evaluating covert channels. I have not read many security papers.

→ BW goes as high as 1.2 MB/s with 2L% error rate.
And a BW of 600 Kb/s with error rate of 1%.

Practical Attack 1: square-and-multiply exponentiation

→ Goal: To find the private key.
→ Depending on the bits of the key, the victim either does square → reduce (for bit "0"). & square → reduce → multiply → reduce (for bit "1").
→ Observing this access pattern, the attacker can get the key.

problem: identify cache set that holds victim code.
↳ again uses access pattern of all cache sets and tries to correlate it with the victim's access pattern

(X) In case of large # of matches → decision of choose the cache set is left to the user — manual decision.

Practical Attack 2: Sliding-window exponentiation

→ Identification of cache set containing victim code
— identify cache set containing multiplication code.
— Scan all the cache sets and collects trace patterns.
- Filter out irrelevant trace patterns.
↳ statistical predictions of multiplier usage patterns.
- recover multiplier usage patter & calculate exponent.

(X) → Identifying the cache set's of interest takes about 37 mins on server systems.
Also involves manual processing to completely recover key.