Take A Way: Exploring the Security Implications of AMD's Cache Way Predictors

Spectredown (**# 4**)
Prajeeth.S
(190050117)

I. Summary

The authors have reverse engineered AMD's L1D Cache way predictor, resulting in two new attacks - COLLIDE + PROBE and LOAD + RELOAD. They demonstrate a covert channel with improved transmission rate over the state-of-the-art, a key recovery attack on AES T-tables and an entropy reducing attack on ASLR of the Linux kernel. Finally, they provide some countermeasures for mitigation.

II. Details

- The way predictors lead to only one comparison of the cache tag per set. Every cache line is associated with a μTag, an unknown hash of its virtual address. On a memory load, only the way with the same μTag as the hash of the incoming address, leading to lesser power consumption. It disallows two virtual addresses with same μTag in the same set, causing eviction of one due to the access of the other.
- A pool of virtual addresses is partitioned into sets, each containing addresses with the same μTag. The hash function is assumed to be linear and is recovered using the sets constructed above.
- COLLIDE + PROBE: Exploit the fact that two virtual addresses in the same set can't have the same μTag. The attacker and the victim must share the same logical core. To monitor victim's access to a virtual address v, the attacker chooses another address v' such that $\mu_v = \mu'_v$. First, the attacker accesses v', updating the μTag. Next, the victim is scheduled to perform its operations. Finally, if the victim had accessed v, v' would've been evicted, leading to an increased access time for the attacker.
- LOAD + RELOAD: Exploits the fact that two virtual addresses of the same set can't have the same physical tag. To attack the memory location with virtual address v, the attacker chooses another virtual address v' that has the same physical tag as v, and loads it. Then, the victim is scheduled to perform its operations. Finally, if the victim had accessed v, v' would've been evicted, leading to an increased access time for the attacker.
- The authors provide a few countermeasures - Dynamic disabling of the Way predictor(in case of continuous misses), keyed hash function (uses context-dependent keys which are updated regularly), State flushing (flushing the way predictor on a context switch), and finally, a purely software based mitigation(mapping secret data n times, such that it can accessed from n different virtual addresses with different μTags).

III. Strengths

- The attacks require only virtual addresses to monitor the victim, which are easier to obtain than the physical addresses(as it does not involve any address translation)
- Collide+Probe provides cache line level monitoring of the victim's access without using shared memory.
- The probe phases of both the attacks are highly efficient since it might require at most an L2 cache access to detect a miss.
- The Collide+Probe attack is used in a high transmission rate covert channel, significantly better than the state-of-the-art values.
- Load + Reload attack can be used to perform cross thread attacks.

IV. Weaknesses

- High-precision timers are required to distinguish between an L1 hit and a miss, which the rdtsc instruction of AMD is unable to provide.
- Collide + Probe requires the attacker and victim to share the same logical core, while the load + reload attack requires shared memory.

V. Extensions

As way predictors make their way into Intel processors, this work can be extended to them as well.

Take a way : Exploring the security implications of AMD's Cache way predictors

→ AMD introduced "Way Predictors" for L1D cache to predict in which cache way a certain addr. is located.

→ This paper reverse engineers this way predictor in micro archs. from 2011 - 2019

Two attacks - Collide + Probe → attacker can monitor victim's memory access without PA/ shared memory.

- Load + Reload.

obtain highly accurate memory-access traces of victim on same core

→ Demonstrate a covert channel, which is also used in a Spectre attack

→ Key Recovery attack

→ Also propose Countermeasures

AMD has always focused on non inclusive / exclusive L1

AMD uses an L1D cache way predictor starting from their bulldozer microarch

Computes a "µTag" using an unknown hash func. on the VA.

Using this, a lookup on prediction table gives the cache way.

reduced power consumption ⇐ So, only one line is checked per set

Claim: Hash func re-engineered - found

Collide + Probe ⇒ µTag collisions of VA

↳ no need for shared mem.

↳ no need for knowledge of physical address.

Load + Reload: exploit the fact that a physical mem. location can reside only once in the L1D cache.

rdtsc - provides unprivileged access to a model-specific register returning the current cycle count.

AMD had a 1-cycle res. until Zen microarch
After, 20-35 cyc. ⇒ It is very difficult to observe one-time events, of shorter duration

APERF - improved acc., but accessed in kernel mode

AMD introduced way predictor, that gives only one check/set, to reduce power consumption

$\mu Tag = f(VA)$ → can be known only from VA
↑ unknown hash
⇒ CPU doesn't have to wait for a TLB lookup. If there is no match for μTag, then early miss and L2 can be issued.

Creating Sets: Get a pool V of virtual addr. that map to the same cache set. [bits 6-11]

$S_0 = \{v_0\}$

for random $v_x \in V$, access $v_x, v_{s0}, v_x, v_{s1}, \ldots, v_x, v_{sn}$

v_{si} = some virtual addr. in the set S_i,

n = number of sets formed so far.

If there is an high access time for v_x after accessing v_{si}, add v_x to S_i.
Else, create $S_{n+1} = \{v_x\}$.

Every VA in the same set gives the same hash. recovered 256 sets (2^8) ⇒ need to know the 8 bits of VA that map to these sets

Assumption that 'linear' hash function.

$h(va)$ = linear combination using XORs.

$$Y_s = a_{12} X_{12} \oplus \ldots \oplus a_{b-1} X_{b-1}$$

Y_s → a common value for all addresses in the set s.

0-5 - line offset
6-11 - set
] cant be used for μTag

Solving this yields an "unordered" set of bits that form the hash.

Two sibling threads → competitively shared
→ statically partitioned.

If data structures of way predictor were competitively shared, one thread can influence another. But, in experiments, no such collision was found. ⇒ Use of per thread info for selecting data entry

Collide + Probe: If address A_1 accessed, μTag computed & updated.
If another address with same μtag accessed, collision occurs ⇒ fetch from L2D cache.

↳ Attacker - unprivileged, on same logical core, can also force victim's code execution

Collide: Virtual addr. v, v' s.t $\mu_v = \mu_{v'}$. Attacker accesses v'. ⇒ μTag updated

Probe: Access v'. If time ≥ threshold, victim has accessed v.

Schedule victim: victim is scheduled to access v.

Load + Reload: Attacker can execute unprivileged code, need not be the same logical CPU thread.
Attack on VA v, shared b/w attacker & victim

Load: Find VA v' that has same physical tag as v.
⇒ v inaccessible from L1D.

Scheduling victim: Access v. ⇒ L1D miss, ⇒ accesses value from L2, invalidating v'.

Reload: Access v'.

Counter measures:
↳ Dynamic Way predictor - disable if lot of misses
↳ Keyed hash functions.
↳ State flushing
↳ Uniformly distributed collisions