Conflict based attack on last level randomized cache

Ippili Sidhartha Kumar
Dept. of Electrical Engineering
IIT Bombay
sidhartha@ee.iitb.ac.in

Rajiv Vaidyanathan Natarajan
Dept. of Computer Science & Engineering
IIT Bombay
203059003@iitb.ac.in

***Abstract*—Shared caches are vulnerable to side-channel attacks and also vulnerable to be used as a covert-channel for information exchange between two processes. To secure the caches from these conflict based attacks, randomized caches have been proposed like CEASER, CEASER-S etc which perform random mapping of cache sets as well as support access based remapping of sets. One previous work concluded that instead of access based remapping, eviction based remapping should be done. In this work, we demonstrate that such a design is also vulnerable to side-channel attacks by exploiting the knowledge of replacement policy (LRU) to form eviction sets using minimal number of evictions. The proposed algorithm has $O(n)$ complexity and hence as effective as Group Elimination Method is for access based remapping. We show that only with highly idealistic assumptions for the adversary, a prime+probe attack is possible on a simple victim trace. We also analyse the extent of success of prime+probe attack for various configurations obtained by varying the remap rate and number of partitions.**

***Index Terms*—randomized caches, side-channel attack**

I. Introduction

Caches improve performance by exploiting the spatial and temporal locality of demand request by the processor. They hide the latency for a significant portion of the demand. Since they modify the time required to access a particular memory location, it is possible to modulate some information on cache access pattern. Many previous works have proposed various side-channel attacks like flush+reload [1], prime+probe [2] etc. which exploit the idea of detectable access time of a particular address location to identify whether a victim code has accesses a particular set or not. We focus on conflict based attacks like prime+probe where the knowledge of which addresses map to the same set is required. For a traditional (*n*-way set associative) cache, the mapping of address bits to physical set number is deterministic, constant and can be calculated by an adversary with enough knowledge of the cache's micro-architecture. Many previous works have reverse engineered this mapping. Some features of system software like huge page support further ease the problem of determining the address mapping. Since this enables the adversary to deploy carefully planned eviction sets to perform eviction based attacks, some works have proposed randomized caches to mitigate the ease of identification of the mapping.

Randomized caches like CEASER and CEASER-S use a random mapping between line addresses and the physical set number. This forces the attacker to create eviction sets using some complex eviction set forming algorithms which require significantly high number of accesses and hence become useless once the entries are remapped. The frequency of remapping is controller by the remap rate and setting a high enough value reduces the effectiveness of the eviction set as it will be destroyed as sets are remapped.

Several works have proposed efficient algorithms to generate eviction sets within the remap period so that they can be used for conflict-based attack without triggering remap. However no practical attack has been demonstrated using such algorithms. In this work, we use such an algorithm that efficiently generates eviction sets for CEASER variant that uses eviction based remapping. We briefly establish why eviction based remap is better than access based remap and further conclude eviction based remapping is as vulnerable as access based remapping. Then we attempt to mount prime+probe attack by assuming an idealistic attacker in order to establish an upper bound on the information leakage by the attacker.

II. Background

A. Overview of Randomized caches

CEASER randomizes the mapping between the line address and the physical set number. It uses encrypted line address to access cache and periodically changes the encryption key to realise the randomized mapping. The remapping of sets is controlled by the *Remap Rate* (R). We note that R is a security parameter and a higher R provides higher security. Additionally, a higher R means more frequent remapping and consequently higher performance/energy overhead.

It should also be noted that, any algorithm that can perform conflict based attack within the remap period will be able to successfully leak information. To achieve this, various existing works have proposed fast eviction set forming algorithms like *Group Elimination Method* (GEM). It has been established by previous works that, to protect CEASER against such attacks, R needs to be set to prohibitively large values. We also note that all the proposed attacks work on CEASER that count the number of accesses for remapping.

Previous works [3] have proposed efficient algorithms for generating eviction sets that can form eviction sets with $O(n)$ complexity in number of accesses. This leaves a possibility of an adversary using the eviction sets for attack before the remap happens. Note that these attacks aim to minimize the number of accesses because of the fact that remapping happens based on the number of cache accesses.

CEASER-S extends CEASER by integrating it skewed caches. The ways of the cache are divided into partitions and

each partition is governed by it's own set of keys. When a new block has to be inserted, a partition is first picked randomly, and then the corresponding key is used to calculate the encrypted line address. The security claim made by CEASER-S depends on the fact that two lines which conflict on one partition may not conflict on other partition(s), hence there is a notion of hard conflict and partial conflict. Because of the fact that the partition is picked randomly, the effectiveness of the eviction sets reduces significantly as compared to CEASER.

B. Need for eviction based remapping in CEASER/CEASER-S

Previous work [5] has proposed the measurement of remap period by evictions rather than accesses because the probability of successfully finding an eviction set is closely related to the number of evictions allowed between remaps. They also conclude analytically that remapping by counting LLC evictions is much more efficient than counting LLC accesses because the LLC miss rate of normal applications is much lower than attacks.

Access based remapping is also prone to denial-of-service attacks. By just making random accesses, an adversary can induce frequent remapping of sets and hence performance of victim's process will be affected. When the latency of CEASER's encryption unit is high, the impact of DoS attack will be further magnified since the encryption unit is in the critical path of cache access mechanism.

III. Attack for eviction-based remapping

As we saw in previous section, a recent work had proposed the use of eviction based remapping. We now attempt to develop an algorithm for eviction set formation that minimizes the number of evictions. We note that this algorithm assumes LRU replacement policy.

(1) We start by accessing an array of address lines denoted by **A**, **B** ... **L**. We assume the cache state to be as depicted in Fig. 1 after the array access. **(2)** Let **M** be another line which maps to the same set as **A** which is part of the access pattern as shown. Now, accessing **M** will evict **A** as per the LRU policy. **(3)** After accessing **M**, we re-access the previous lines. While doing so, we find that **A** was not in cache hence **A** is added to the eviction set. As a consequence of accessing **A**, **A** has now evicted **B** which again denotes that **B** should also be added to the eviction set. **(4)** Similarly, **D** is also added to the eviction set.

We observe from the above example that the number of evictions required to form the eviction set has $O(n)$ complexity. Hence this approach has the same threat level as the algorithm for access based remapping (GEM).

IV. Experimentation

All the analysis done in this work is implemented on ChampSim, which is a trace based simulator. Fig. 2 gives an overview of the simulation setup. Since ChampSim simulates a trace, we can only provide the victim trace as input and the attack has to be integrated inside the simulator. Hence the attacker subroutines like prime/probe methods, eviction set creation and adversarial cache line insertion were written into the simulator source.

TABLE I: ChampSim Configuration

L1D and L2 cache	1 set, 1 way
LLC	2048 sets, 16 way LRU replacement policy CEASER with APLR = 100

It should be noted that the default ChampSim simulator does not have an inclusive cache hierarchy which would be necessary for a simpler attack strategy. Additionally, to perform a side-channel attack on LLC, we need to make almost all of our victim's access reach the LLC. To emulate the inclusiveness and to ensure that victim accesses reach the LLC, we reduce the sizes of L1D and L2 caches to very small values.

We note that integrating the adversary code into the simulator does not accurately reflect the realistic scenario but actually models a more idealistic scenario. This actually helps us to realise a worst case or upper bound on the information leakage that is possible. Additionally, assuming an idealistic attacker would also guarantee that the security comes solely from the intended micro-architecture and not from the underlying noise. Further, various works [5] [6] have also assumed a similar level of access for their attacker model.

A. Representational victim code

Previous works have targeted insecure versions of cryptographic algorithms like AES or RSA which have one central idea that the victim code makes secret key based accesses to memory locations. The aim of the attacker in this setup has always been to identify which sets of the cache are accessed and based on that information, obtain the victim's secret data. We observed that access pattern of these traces is complex and also the memory footprint of these traces is large, making it difficult to analyse the security or feasibility of eviction based attack on randomized caches. To this end, we tried to simplify our victim model by constructing a very rudimentary version of the square/multiply routine employed by RSA.

We still preserve the central idea that the victim makes data-dependent access but aim to reduce the memory footprint of the victim trace. We start with the code shown in listing (initial victim code). After running this trace, we observe that the location corresponding to iterator `i` was going into the cache even with the `register` declaration. From the corresponding LLC access pattern (a snapshot is shown in Fig. 3), we observed that the victim access pattern is not ideal for making a distinction between bit 0 and bit 1 of victim's secret key. To remove this obstacle, we use an unrolled version of the loop as shown in listing (final victim code).

B. Access interval calibration

It is important that the frequency of prime and probe functions match the set access interval of victim. To achieve this we first observe the interval (in terms of cpu cycles) between

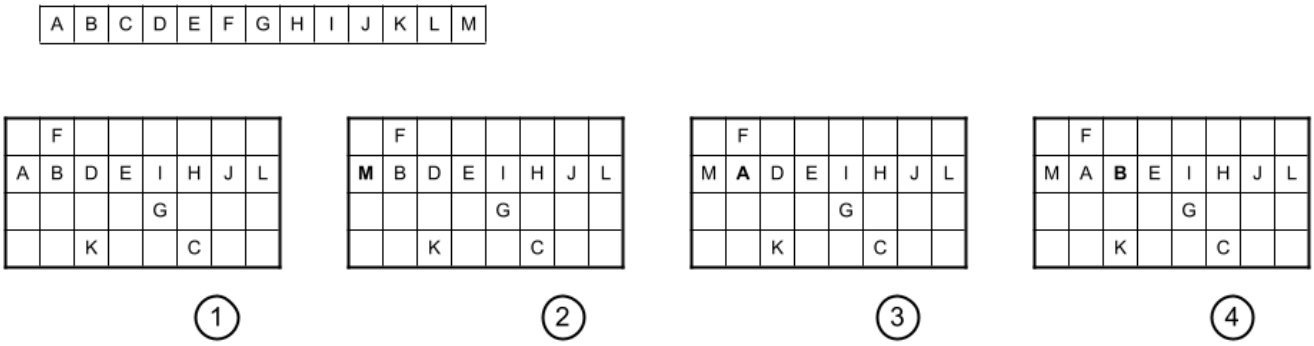

Fig. 1: Example: Eviction set formation

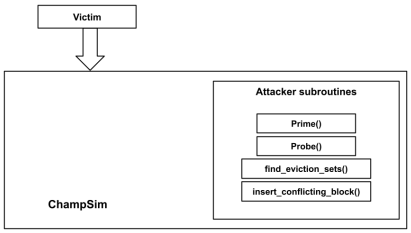

Fig. 2: Simulation setup: ChampSim with attack functions built into the source

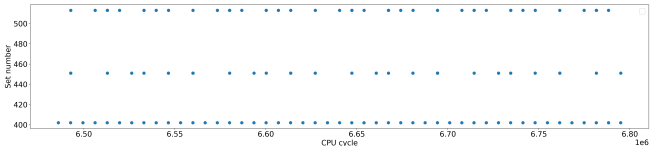

Fig. 3: Temporal set access pattern of victim obtained directly from simulator (traditional LLC)

two consecutive LLC accesses from within the simulator. A plot of access interval for a small window of simulator run is shown in Fig. 5. From the plot, we decide to use an interval of 4200 cycles.

It should be noted that identifying the access interval from within the simulator does not make the strategy idealistic. This is because the sources of many potentially vulnerable codes are publicly available and they can be run on a personal machine which is architecturally similar to the target machine and figure out the access interval through some debugging or some other form of side channel attack.

C. Orchestrating evictions inside simulator

The eviction set forming subroutine is implemented inside the function that is responsible for finding the a victim block in a cache set when the simulator needs to evict a block to accommodate the newly inserted block. Because of this, we need to orchestrate evictions by accessing addresses that have high probability of conflicting with existing lines. This was initially done with the help of a trace that was purposely built for thrashing the LLC and the initial idea was to append the victim code after enough iterations of the thrashing code.

Since the eviction sets will be destroyed due to remapping as well as conflict with victim accesses, it will be necessary to orchestrate more evictions later. Because of this it was decided to generate this thrashing access pattern from within the simulator. Secondly, since generating evictions is part of attacker anyways, it makes sense to separate this from victim's trace. For this, we created a function `insert_block()` which is identical to the CACHE class's fill method but does not account for the timing model of the different structures like MSHR read, interconnect latency etc.

1) Access pattern for orchestrating evictions: As we know CEASER randomizes the mapping between of set-index bits to physical set number, it is most likely that an access pattern that would thrash a traditional cache will be as good as a random access pattern. Therefore, we call `insert_block()` with randomly generated addresses.

2) Frequency of block insertion by attacker: After establishing an access pattern, we determined the frequency of inserting cache blocks by the attacker. As we are interested in LLC, we note the fact that an average program running on a core can make one LLC access (approximately) every 100 cycles. Empirically, we can also say that a highly agile adversarial code can make an LLC access every 50 cycles. Considering this, we keep the frequency of block insertion at 1 in every 100 cycles.

D. Prime+Probe using simulator information

Using simulator data, we obtained which sets of LLC are accessed by victim code. To prevent remapping of these sets, we selectively prime and probe only these locations. As shown in Fig. 6, the probed output matches the victim's access pattern. We also note that, while this attack was successful, the assumption that attacker can identify which sets to prime/probe is too idealistic and unreal. Hence we don't consider this as a successful attack. To obviate this assumption, a batch-like prime/probe was also considered but it was unable to catch the victim's access pattern.

```
for(register int i=0;i<10;i++) {
    if(key[i]==1) y->val4+=1;
    else x->val4+=1;
    asm volatile ("mfence");
    for(register int j=0;j<2000;j++)
        asm ("nop");
    asm volatile ("mfence");
}
```

(a) Initial simplified victim code

```
while (b<a) {
    y->val4+=1; // represents 0
    sleep();
    x->val4+=1;  // represents 1
    sleep();
    . . .
}
```

(b) Further simplified victim code

Fig. 4: Realizing a simple victim trace

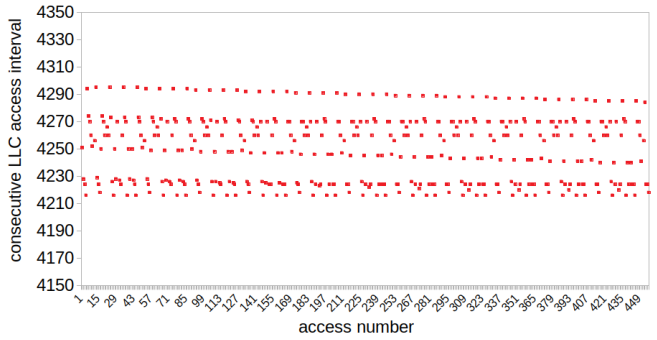

Fig. 5: Interval of consecutive LLC set access for a snippet of victim trace

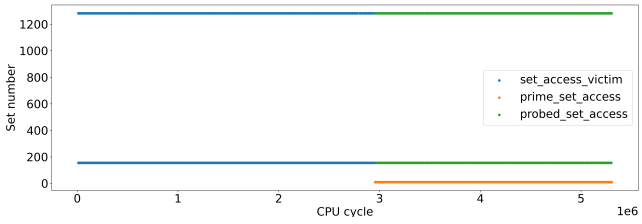

Fig. 6: Temporal set access pattern obtained directly from simulator before prime+probe, probed result and prime cal

V. Results and Discussion

The security of any cache design is also determined by the number of eviction sets that can be formed. Hence, we identify the maximum number of eviction sets generated by the algorithm for various remap rates. Since this didn't require any victim trace, we use a trace which generates a thrashing access pattern as input trace for simulator and do not insert block from within the simulator. It is worth noting that in Fig. 7 the maximum number of eviction sets is higher for CEASER-S, but that does not mean skewed-CEASER is less secure than CEASER, because the effectiveness of these eviction sets formed is lower than that in case of CEASER. This is because of the additional randomization in selecting the partition for an incoming block.

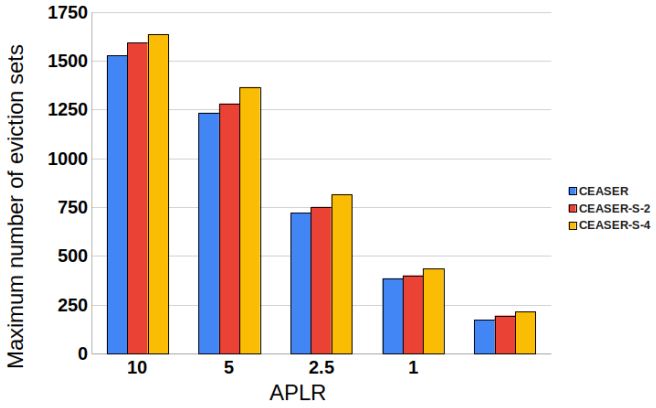

Fig. 7: Maximum number of eviction sets for various APLR values

Even though we are unable to perform a successful retrieval of secret key, we note that the performance of victim program can be hampered by deliberately orchestrating evictions using the collected eviction sets. This gives an adversary the opportunity to launch denial-of-service attacks on a randomized cache even in the presence of eviction-based remapping.

VI. Conclusion

While eviction based remapping might be more efficient that access based remapping, we saw that equally efficient eviction set finding mechanisms exist similar to GEM for access based remapping. So there is still a trade-off between security and performance while selecting the remapping logic and we cannot be certain that eviction based remapping is more secure than access based remapping. Further, the effectiveness of eviction sets decreases with increase in partition count and even CEASER with appropriately set remap rate is secure against conflict-based attacks. Also, even though randomized caches don't appear to be vulnerable to practical conflict-based attacks, a motivated adversary can definitely perform a denial-of-service attack by artificially increasing the remapping.

ACKNOWLEDGMENT

The authors of this report would like to thank Prof. Biswa for his valuable guidance and insights.

REFERENCES

[1] Yuval Yarom and Katrina Falkner. 2014. FLUSH+RELOAD: a high resolution, low noise, L3 cache side-channel attack. In Proceedings of the 23rd USENIX conference on Security Symposium (SEC'14). USENIX Association, USA, 719–732.

[2] F. Liu, Y. Yarom, Q. Ge, G. Heiser and R. B. Lee, "Last-Level Cache Side-Channel Attacks are Practical," 2015 IEEE Symposium on Security and Privacy, 2015, pp. 605-622, doi: 10.1109/SP.2015.43.

[3] M. K. Qureshi, "New Attacks and Defense for Encrypted-Address Cache," 2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA), 2019, pp. 360-371.

[4] M. Qureshi, "CEASER: Mitigating Conftit-Based Cache Attacks via Encrypted-Address and Remapping", 51st Annual IEEE/ACM International Symposium on Microarchitecture, Oct 2018.

[5] W. Song, B. Li, Z. Xue, Z. Li, W. Wang and P. Liu, "Randomized Last-Level Caches Are Still Vulnerable to Cache Side-Channel Attacks! But We Can Fix It," 2021 IEEE Symposium on Security and Privacy (SP), 2021, pp. 955-969, doi: 10.1109/SP40001.2021.00050.

[6] Daniel Genkin, William Kosasih, Fangfei Liu, Anna Trikalinou, Thomas Unterluggauer, Yuval Yarom, "CacheFX: A Framework for Evaluating Cache Security" Arxiv, Preprint, https://doi.org/10.48550/arXiv.2201.11377