Disabling Prefetcher to Enhance Cache Side Channels

1st Meet Udeshi
Electrical Engineering Department
Indian Institute of Technology, Bombay

***Abstract*—Cache side channels are well known for being effective in extracting data from modern cryptographic ciphers. Side channel attacks gather data on their own cache hits and cache misses, and can infer the program's memory accesses based on this data. Such attacks assume that the victim program is the only other one making accesses to the shared cache. Any accesses not originating from the victim program lead to false positives which introduces noise in the side channel. Attackers can restrict other programs from running on the core and synchronize the attack with execution of relevant section of code of the victim program. However, they cannot directly switch off hardware units like Prefetchers which may generate accesses to the cache. This work describes a method in which the prefetcher is prevented from generating any accesses using an attack tailored to disable its function.**

***Index Terms*—security, side channels, prefetcher**

I. Introduction

An attacker program using the cache as a side channel tries to force collisions with the victim program by making accesses which alias to the same cache lines [1]. Time taken for subsequent accesses to these cache lines differs and is used to determine whether there was a successful collision or not. This data is further used to infer whether the victim accessed a particular cache line or not, thus leaking data about the data of the program. Different implementations of the side channel look for either a cache hit or a cache miss as a sign of successful collision. The Prime+Probe attack fills the cache lines in a set with data other than that being accessed by the victim. Any access by the victim to that set will cause attacke's data to be evicted, which will show up during the Probe step as a cache miss [1]. Similarly, the Flush+Reload attack looks for a cache hit to the same data as the victim. A cache hit in the Reload step is infered as successful collision [2].

The attacker assumes a scenario where only the victim is making memory accesses, hence is able to deduce the memory access pattern. If there is another program or hardware making memory accesses, they will surely interfere with the side channel. After obtaining a successful collision, there is no way for the attacker to distinguish whether the source of this collision was truly the victim. Fuchs et al [3] introduce a Disruptive Prefetcher which generates spurious memory accesses, making the victim's accesses indistinguishable for any attacker.

This paper focuses on a way to disable the prefetcher and significantly reduce the number of generated prefetches. With a separate attacker focusing on disabling the prefetcher, it becomes extremely unlikely for the side channel attacker to see a collision with a prefetcher generated access. This enhances the side channel and can enable faster and better data retrieval. The attack implementation has been designed specific to a Stride prefetcher [4]. However, the implementation can be used as-is or easily extended to apply to any PC-indexed prefetcher table.

A Stride Prefetcher tries to identify certain load instructions which have a memory access pattern

II. Attack Vectors

- Prefetcher will be disabled when it does not get to the condition of generating prefetches. - They generate prefetch when confidence value crosses threshold. - Confidence value is reset below threshold when a new entry is made to the table. This is exploited by periodically making new entries into the table to evict older entries and prevent them from gaining confidence. - This attack vector allows us to never let victim's load instruction trigger the prefetcher to generate prefetches. - Confidence is decremented when an entry hits in the table and the stride value does not match. Attacker randomises the stride value in every access. - This prevents attacker from generating any prefetches which would be detrimental to the purpose. - Prefetcher is only accessed when there is a cache miss, clflush instruction is used to always generate a cache miss.

III. Attacker Implementation

- To add new entries to Prefetcher table, we need to have a load instruction at new location. - A binary is created which contains large number of load instructions placed at different PC addresses so that after aliasing every entry of the prefetcher table is accessed atleast once.

A. *Full Attacker*

- To make load instructions alias to every entry, their PC addresses need to be carefully controlled. The set indexing bits need to attain every possible value enough times to fill all ways. - Single load is 4 bytes in X86. accompanying clflush is 3 bytes. sequence of 7 bytes placed one after other will skip certain set indexes. A particular set index may have many loads aliasing to it while another may have very few. - To properly control, padding with nop instruction is required. - Describe why 8 byte sequence and sequence length + nop

```
00000000000006ca <attack>:
 6ce:  8b 58 36        mov     0x36(%rax),%ebx
 6d1:  90              nop
 6d2:  0f ae 78 36     clflush 0x36(%rax)
 6d6:  8b 58 08        mov     0x8(%rax),%ebx
 6d9:  90              nop
 6da:  0f ae 78 08     clflush 0x8(%rax)
 6de:  8b 58 3f        mov     0x3f(%rax),%ebx
 6e1:  90              nop
 6e2:  0f ae 78 3f     clflush 0x3f(%rax)
 6e6:  8b 58 38        mov     0x38(%rax),%ebx
 6e9:  90              nop
 6ea:  0f ae 78 38     clflush 0x38(%rax)
 6ee:  8b 58 20        mov     0x20(%rax),%ebx
 6f1:  90              nop
```

Listing 1. Full Attacker disassembly: load misses at different PCs

- This is called the full attacker because it targets the entire prefetcher table. - The full attacker takes time to run a single iteration of the attack because of the repeated cache misses and it contains 256 loads. - It is possible (and also observed) that some load addresses of the victim can retrain the prefetcher while one iteration is running and generate prefetches. - These retrained load instructions will only be evicted in the next iteration.

B. Targeted Attacker

- A faster implementation is required which can quickly evict notorious loads of the victim. - If such loads are identified in advance, an attacker which targets the sets of only such loads will be required. - Loads can be identified by looking at memory access patterns of the victim program by running simulations with multple random inputs. - From the full attacker we can leave relevant load instructions which alias with the targeted PCs, filter out others by replacing them with nop. - for targeting 2 PCs the number of load instructions required reduces from 256 to 16

```
00000000000006ca <attack>:
          ...
 6d9:  90              nop
 6da:  8b 58 0f        mov     0xf(%rax),%ebx
 6dd:  0f ae 78 0f     clflush 0xf(%rax)
 6e1:  90              nop
 6e2:  90              nop
 6e3:  90              nop
 6e4:  8b 58 3c        mov     0x3c(%rax),%ebx
 6e7:  0f ae 78 3c     clflush 0x3c(%rax)
 6eb:  90              nop
 6ec:  90              nop
        <nop slide> ...
 6f7:  90              nop
 6f8:  8b 58 2f        mov     0x2f(%rax),%ebx
 6fb:  0f ae 78 2f     clflush 0x2f(%rax)
 6ff:  90              nop
          ...
```

Listing 2. Targeted attacker disassembly: loads at aliased PCs

- The nop slides although important do not add any significant delay compared to the cache misses of load instructions.

IV. Simulation

- Gem5 X86 simulator - Two OoO cores with shared L2 cache - Stride prefetcher attached to L2 cache 16x4 - Victim

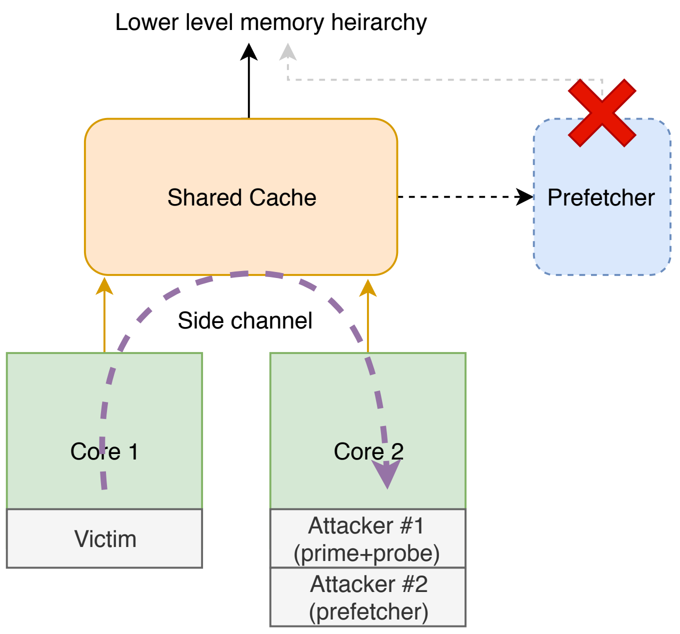

Fig. 1. Setup of attack to disable prefetcher from generating memory accesses

Simulator	gem5 X86
Cores	2
L1 Icache	32K 8-way
L1 Dcache	32K 8-way
L2 cache	256K 16-way shared between cores
L2 prefetcher	Stride 64-entry 4-way, confthresh 4

TABLE I
Simulation Setup

program runs on core 1 and attacker runs on core 2 - Simulator measures number of prefetches issued, average confidence, hits and misses to table - These are measured for every 1 million instructions

V. Results

- Benchmarks show full attacker is effective against some types of memory access patterns while others are not reduced much.

- AES victim is implemented which encrypts random data using AES implementation of openssl 0.9.x (uses libcrypto) - Simulation of only victim helps identify two load addresses which generate most of the prefetches. - Targeted attacker is tailored to those load PCs - Full attacker is somewhat effective at reducing prefetches is some phases, but targeted attacker is extremely effective.

- The targeted attacker and full attacker are able to equally reduce confidence of prefetcher

- Number of hits are significantly lower for targeted

DCPT prefetcher (hwpf)

- Same implementation works very well for DCPT. - Full attacker works better, why ?

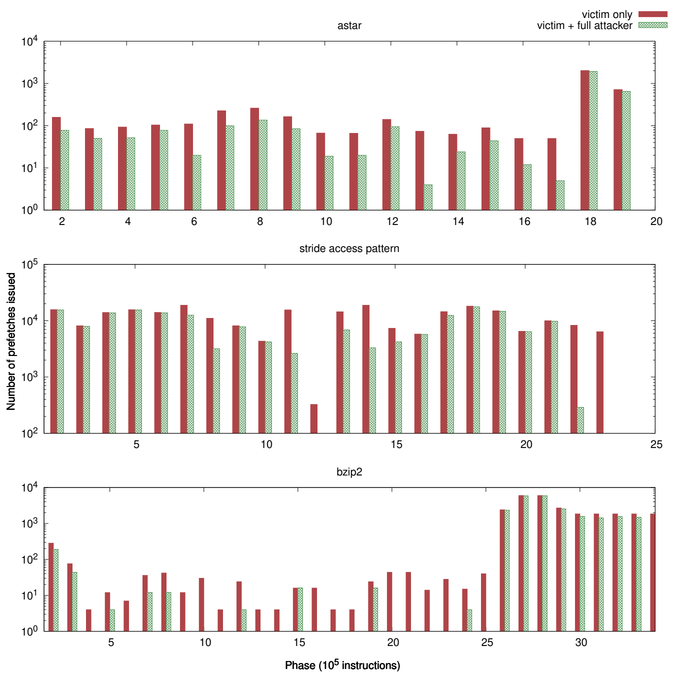

Fig. 2. Number of prefetches issued on different benchmarks

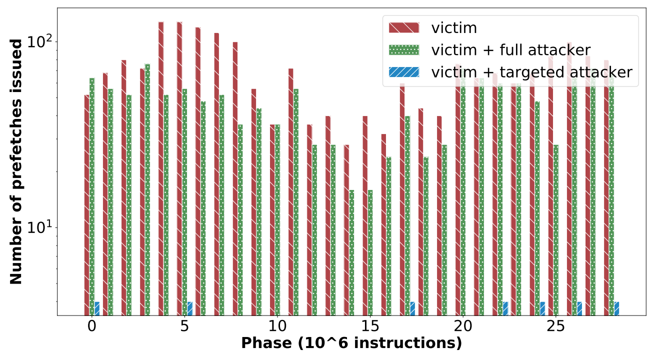

Fig. 4. Comparision of number of prefetches issued by AES program

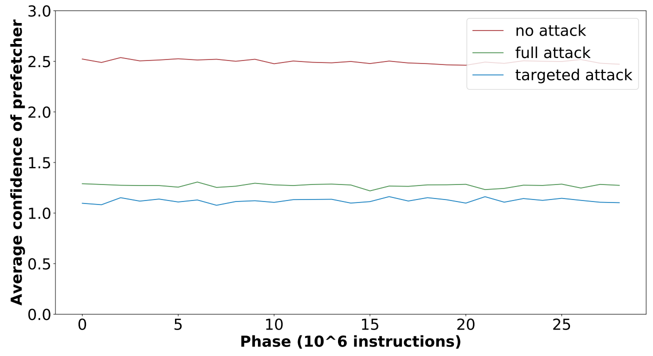

Fig. 5. Comparision of average confidence of prefetcher with AES program

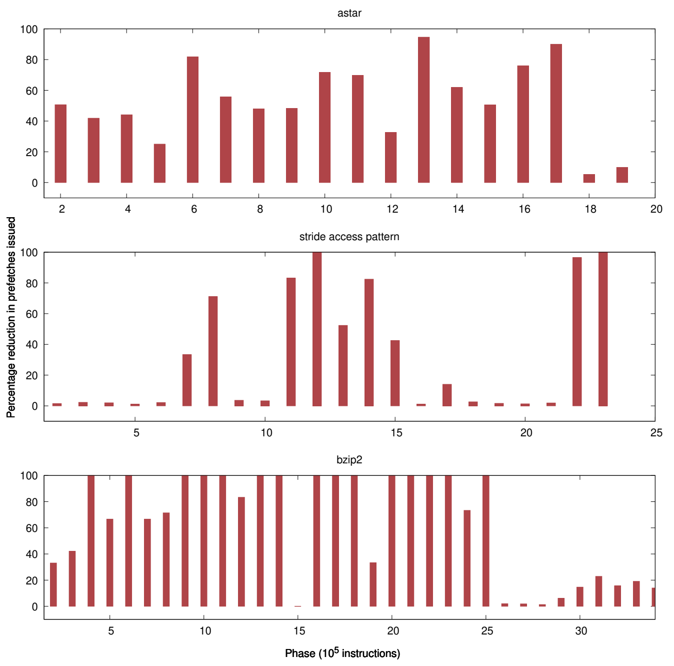

Fig. 3. Percentage reduction in number of prefetches

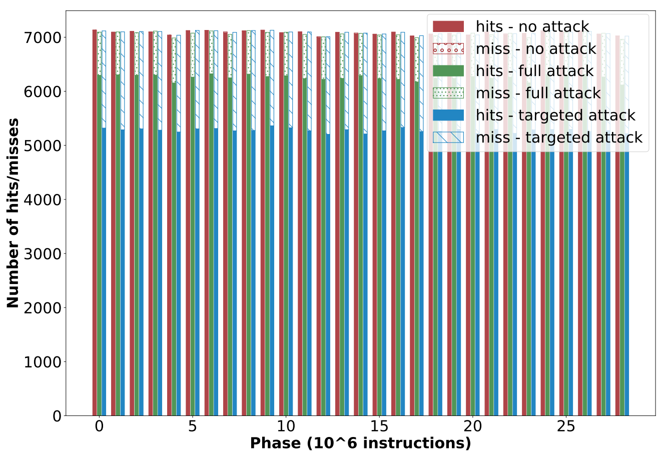

Fig. 6. Comparision of prefetch table hit and miss count by AES program

VI. Conclusion

References

[1] D. A. Osvik, A. Shamir, and E. Tromer, “Cache attacks and countermeasures: the case of aes,” in *Cryptographers’ track at the RSA conference*, pp. 1–20, Springer, 2006.

[2] C. Percival, “Cache missing for fun and profit,” 2005.

[3] A. Fuchs and R. B. Lee, “Disruptive prefetching: impact on side-channel attacks and cache designs,” in *Proceedings of the 8th ACM International Systems and Storage Conference*, p. 14, ACM, 2015.

[4] J. W. C. Fu, J. H. Patel, and B. L. Janssens, “Stride directed prefetching in scalar processors,” in *Proceedings of the 25th Annual International Symposium on Microarchitecture*, MICRO 25, (Los Alamitos, CA, USA), pp. 102–110, IEEE Computer Society Press, 1992.