Reading Assignment: CS6630

Prasanna Bartakke

EE19B106

Adversarial Prefetch: New Cross-Core Cache Side Channel Attacks

Abstract

`PREFETCHW` Accelerates futures writes

Security flaws:

1. Can execute on data with read-only permissions
2. Execution time leaks the target data's coherence state

Cross-core attacks:

1. Prefetch+Prefetch
2. Prefetch+Reload

CPU Cache Architecture and Coherence Protocol

L1, L2: fast, small, private
LLC: slow, large, shared among CPU cores

Inclusive LLC: Data present in private caches necessarily present in LLC
Most Intel processors except the latest Intel Xeon server processors

Cache coherence
To maintain consistency among the copies of a cache line in different private caches.
E.g., **MESI** coherence protocol.

MESI coherence protocol

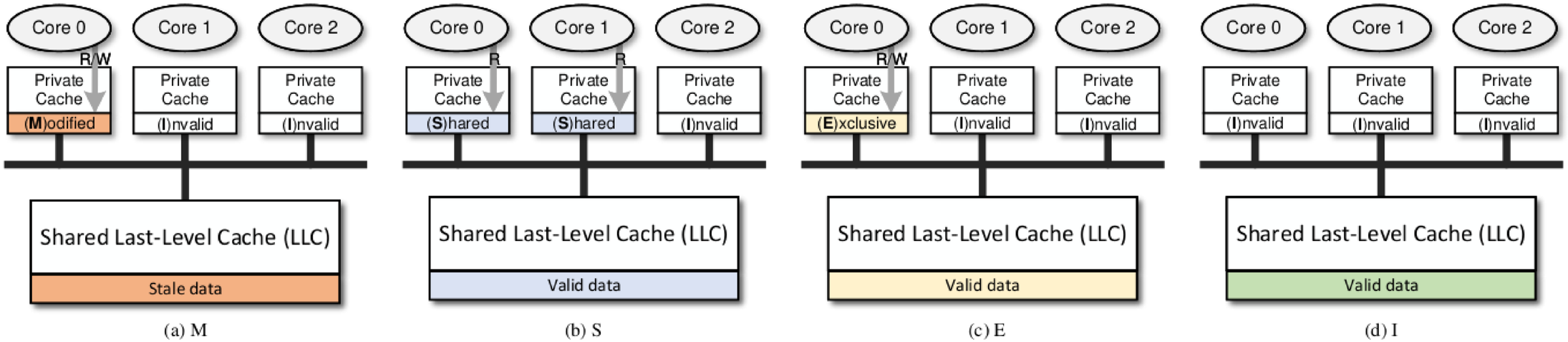
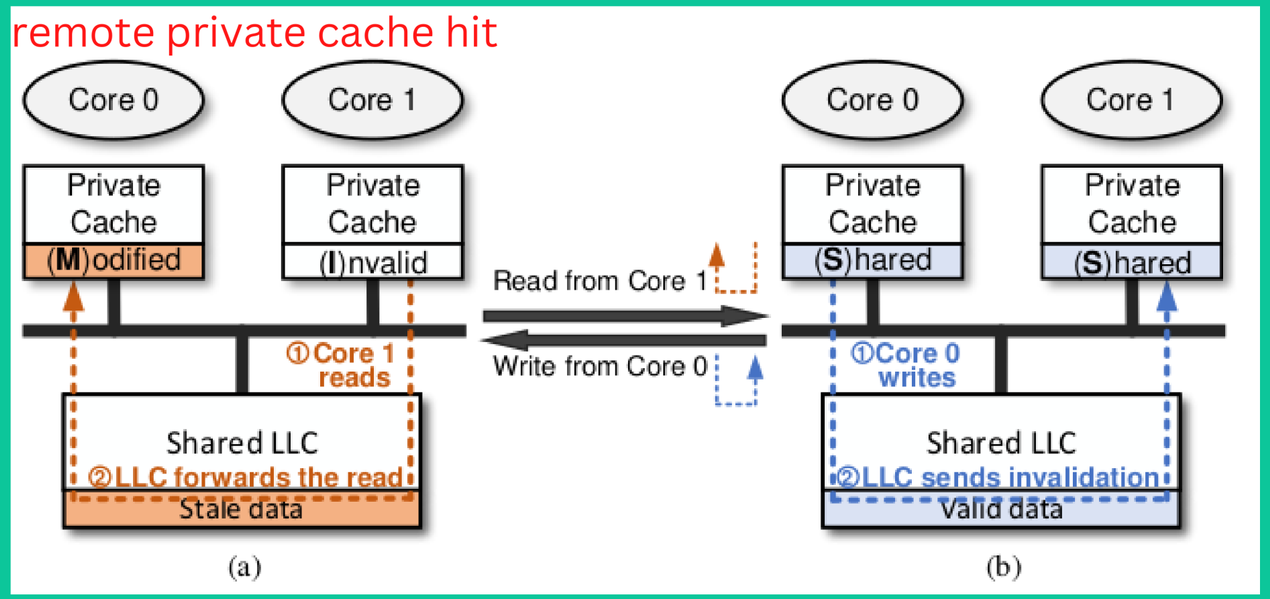


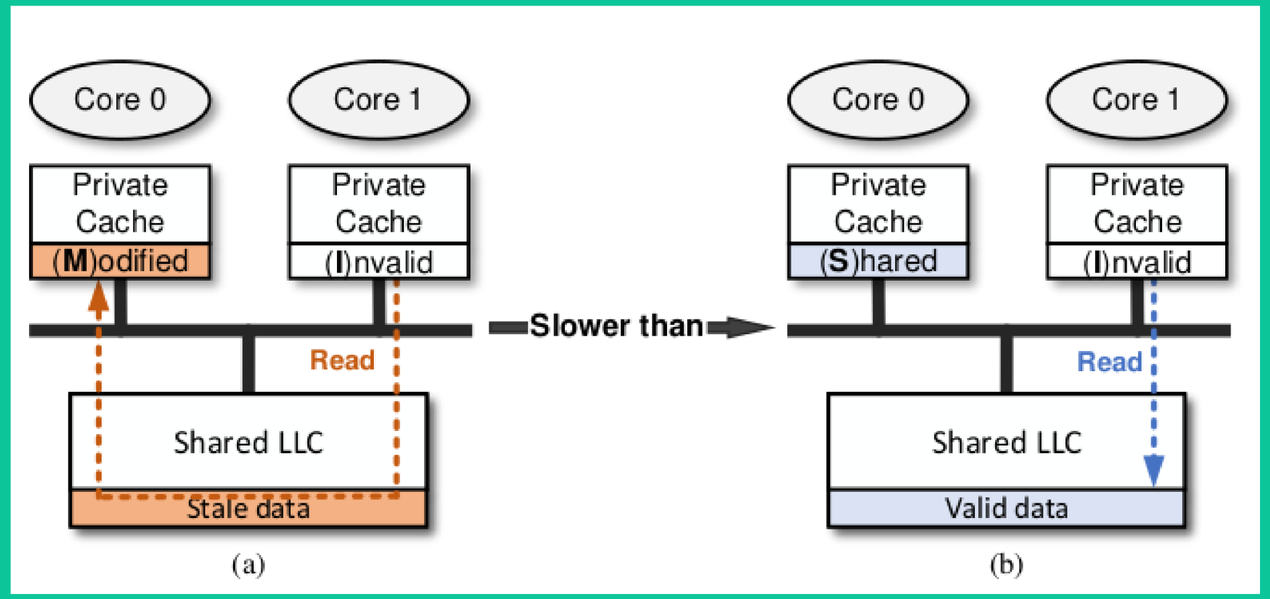
Fig. 1: The four possible states of a private cache line, when using the MESI protocol.

Modified Shared Exclusive Invalid



State transitions

a. M to S when CPU core loads cache line
b. S to M when the CPU core is writing to the cache line



Timing difference

remote private cache hit is slower than LLC hit

Core i7-6700 processor
LLC hit: 60 cycles
remote private cache hit: 90 cycles

Prefetch

1. Technique to boost performance by fetching data before needed and placing closer to the CPU core. E.g. From LLC to the L1 cache.
2. Hardware Prefetch: Implemented in cache hardware, E.g. adjacent cache line prefetcher.
3. Software Prefetch: Explicitly done by the programmer/compiler. Hint to the processor that a memory location is very likely to be accessed soon. E.g. prefetch instructions in x86 processors - PREFETCHT0, PREFETCHT1, PREFETCHT2, PREFETCHNTA, PREFETCHW.

PREFETCHW

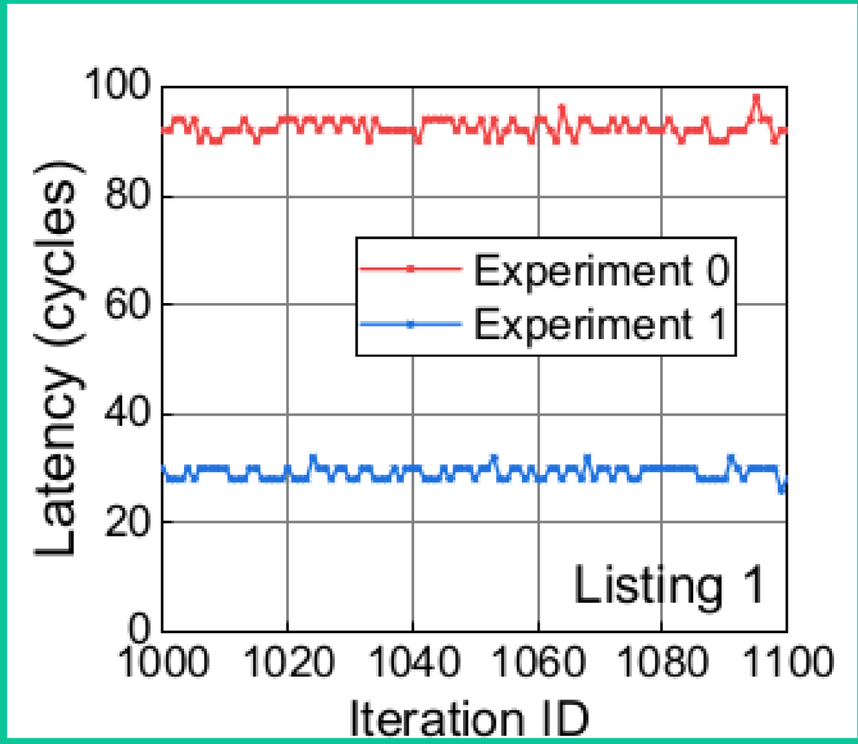
1. Places the target cache line into the L1D cache and sets the coherence state of the cache line to M.
2. Accelerates future writes on target cache line.

```
1  void* thread0 (void* addr_d0, int expt_idx){
2      for(int i = 0; i < 1000000; i++){
3          /* check the experiment index*/
4          if(expt_idx == 0){
5              /* execute prefetchw on d0*/
6              prefetchw(addr_d0);}
7          /*let thread1 execute 1 iteration*/
8          wait_for_thread1();
9      }}
10
11 void* thread1 (void* addr_d0){
12     for(int i = 0; i < 1000000; i++){
13         /*let thread0 execute 1 iteration*/
14         wait_for_thread0();
15         int result = read_and_time(addr_d0);
16     }}
17
18
19 int main() {
20     /* open and map a file as read-only*/
21     int fd = open(FILE_NAME, O_RDONLY);
22     int* addr_d0 = mmap(fd, PROT_READ, ...);
23
24     /*pin thread0 on core0 and start thread0*/
25     /*pin thread1 on core1 and start thread1*/
26     ...
```

Listing 1: The code snippet for verifying Observation 1

Observation 1

PREFETCHW successfully executes on data with read-only permission.

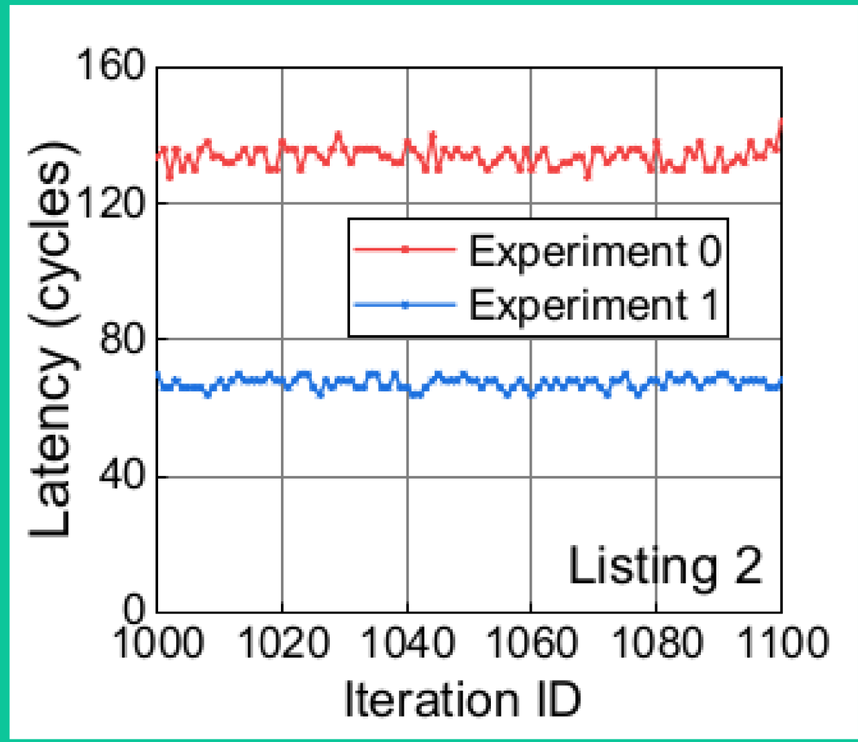


```
1  void* thread0 (void* addr_d0, int expt_idx){
2      for(int i = 0; i < 1000000; i++){
3          /* check the experiment index*/
4          if(expt_idx == 0){
5              read(addr_d0);}
6          /*let thread1 execute 1 iteration*/
7          wait_for_thread1()
8      }}
9
10 void* thread1 (void* addr_d0){
11     for(int i = 0; i < 1000000; i++){
12         /*let thread0 execute 1 iteration*/
13         wait_for_thread0();
14         int t1 = rdtscp(); /* read time stamp*/
15         prefetchw(addr_d0);
16         int result = rdtscp()-t1;
17     }}
18
19 int main() {
20     /* open and map a file as read-only*/
21     int fd = open(FILE_NAME, O_RDONLY);
22     int* addr_d0 = mmap(fd, PROT_READ, ...);
23
24     /*pin thread0 on core0 and start thread0*/
25     /*pin thread1 on core1 and start thread1*/
26     ...
```

Listing 2: The code snippet for verifying Observation 2.

Observation 2

The execution time of PREFETCHW is related to the coherence state of the target cache line.



Prefetch-based covert channel attacks

1. sender and receiver running on different physical cores
2. sender and receiver can share read-only data
3. agree on predefined channel protocols

Algorithm 1: Prefetch+Load Covert Channel

line0: the shared cache line between the sender and receiver
message[n]: the n-bit long message to transfer on the channel
Th0: the timing threshold for distinguishing local and remote private cache hit

Sender Algorithm

```
// Send 1 bit in each iteration.
for i = 0; i < n; i++ do
    sync_with_receiver();
    if message[i] == 1 then
        Prefetch line0;
    else
        Do not prefetch;
```

Receiver Algorithm

```
// Detect 1 bit in each iteration.
for i = 0; i < n; i++ do
    sync_with_sender();
    Access line0 and time the access;
    if access_time > Th0 then
        Received a bit "1";
    else
        Received a bit "0";
```

Prefetch + Load Attack

1. Based on observation 1
2. To send bit 1: PREFETCHW on the shared cache line.
3. To send bit 0: idle
4. Receiver loads the shared cache line and interprets the shared bit by looking at the time required.

Algorithm 2: Prefetch+Prefetch Covert Channel

line0: the shared cache line between the sender and receiver
message[n]: the n-bit long message to transfer on the channel
Th0: the timing threshold on `PREFETCHW` to distinguish M and S states

Sender Algorithm

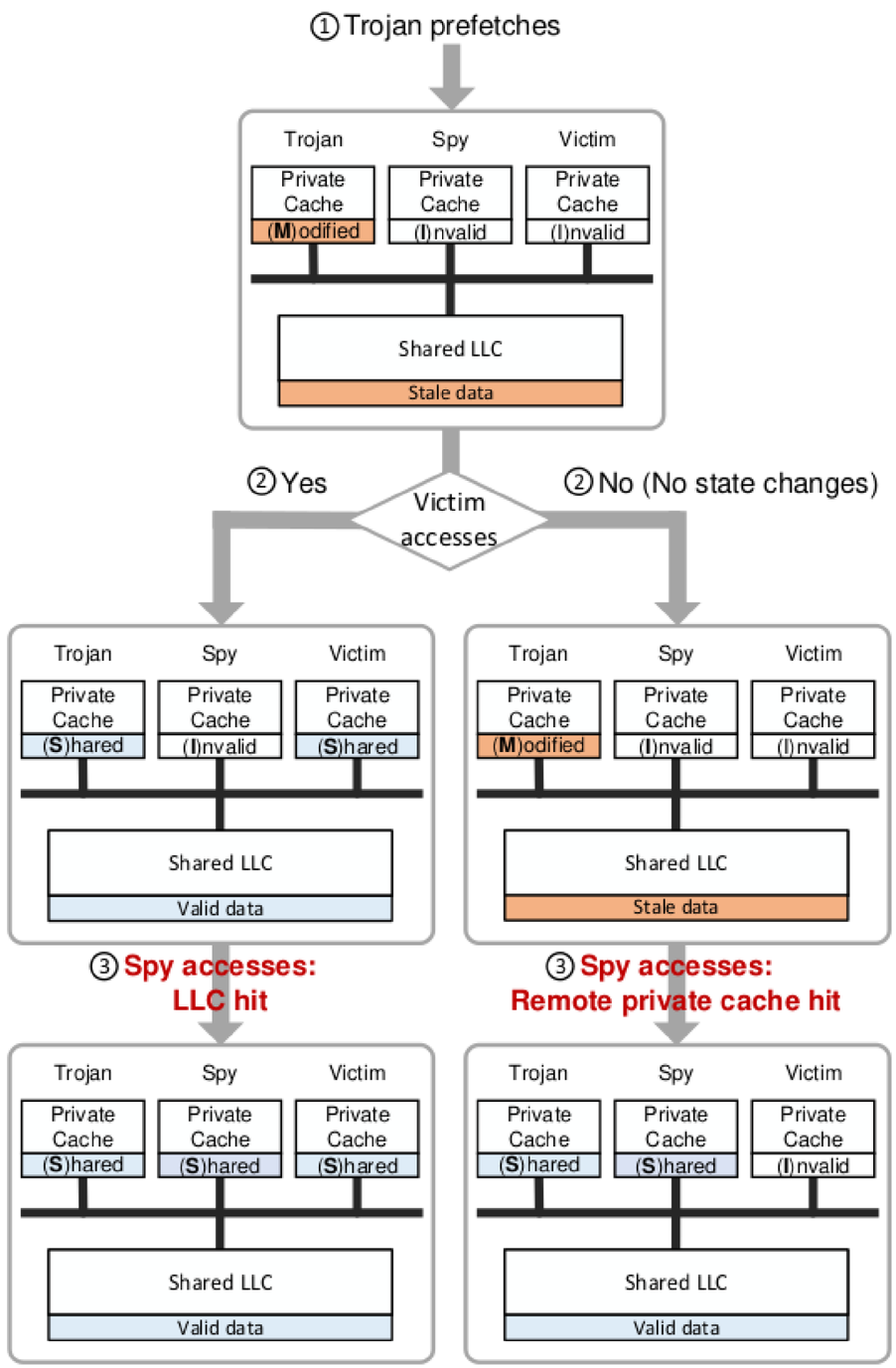
```
// Send 1 bit in each iteration.
for i = 0; i < n; i++ do
    sync_with_receiver();
    if message[i] == 1 then
        Load line0;
    else
        Do not load;
```

Receiver Algorithm

```
// Detect 1 bit in each iteration.
for i = 0; i < n; i++ do
    sync_with_sender();
    Prefetch line0 and time the prefetch;
    if prefetch_time > Th0 then
        Received a bit "1";
    else
        Received a bit "0";
```

Prefetch + Prefetch Attack

1. Based on observation 2
2. To send bit 1: Load shared cache line
3. To send bit 0: idle
4. Receiver executes PREFETCHW on the shared cache line and interprets the shared bit by looking at the time required.



Prefetch-based side channel attacks

Prefetch + Reload Attack

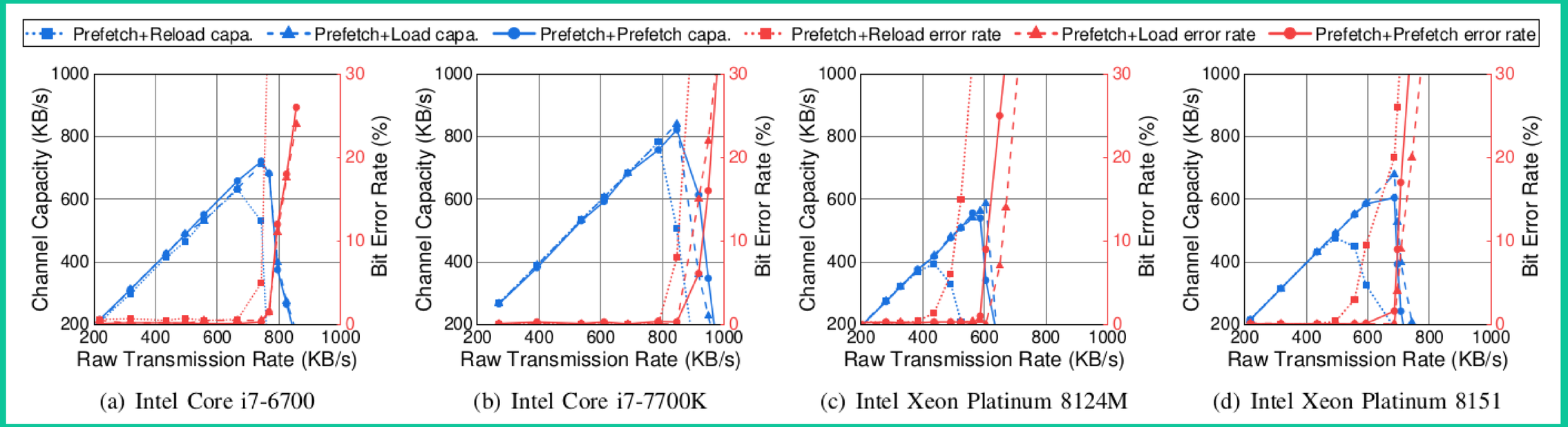
1. Trojan performs PREFETCHW on the target cache line and becomes the exclusive owner of this cache line.
2. The attacker waits for the victim's behaviour: if the victim accesses this cache line, its coherence state will become S, meaning the copy in the LLC is now valid.
3. Spy accesses this cache line and times the access to determine it was a remote private cache hit or an LLC hit. If it was a remote private cache hit, the victim did not access this cache line; otherwise, the victim did access.

Prefetch-based side channel attacks

Prefetch + Prefetch Attack

1. The attacker prefetches the target shared cache line using PREFETCHW, and times this operation to learn whether the victim accessed this cache line in the last iteration.
2. The attacker waits for the victim’s behavior.

Evaluation of Prefetch-Based Covert Channel Attacks



Platform	Desktop processors		Server processors	
	Core i7-6700 (3.4 GHz)	Core i7-7700K (4.2 GHz)	Xeon Platinum 8124M (3.0 GHz)	Xeon Platinum 8151 (3.4 GHz)
Prefetch+Reload	631 KB/s	**782 KB/s**	394 KB/s	476 KB/s
Prefetch+Load	709 KB/s	**840 KB/s**	586 KB/s	680 KB/s
Prefetch+Prefetch	721 KB/s	**822 KB/s**	556 KB/s	605 KB/s

Flush + Reload - 298 KB/s

Flush + Flush - 496 KB/s

Prime + Probe - 438 KB/s

Side Channel Attack on Cryptographic Code

Algorithm 3: Square-and-multiply Exponentiation

Input: base b, modulo m, exponent $e = (e_{n1}...e_0)_2$
Output: $b^e \mod m$

$r \leftarrow 1$
for $i = n-1;\ i >= 0;\ i--$ **do**
    $r \leftarrow r^2 \mod n$
    if $e_i == 1$ **then**
        $r \leftarrow r * b \mod n$

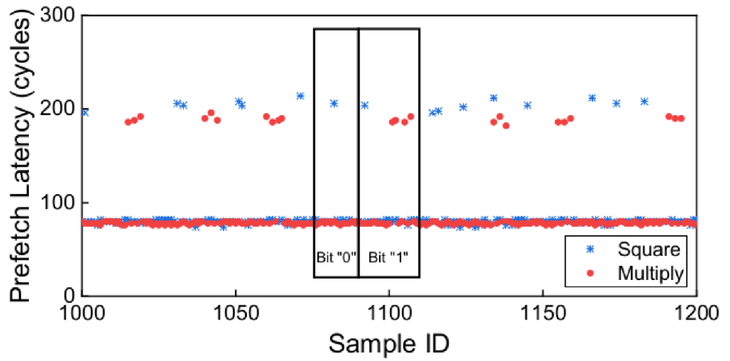


Fig. 7: A segment of the prefetch latencies measured in Prefetch+Prefetch while attacking GnuPG; part of the the exponent e shown here is "111001011001".

Side Channel Attack on Keystroke Timing

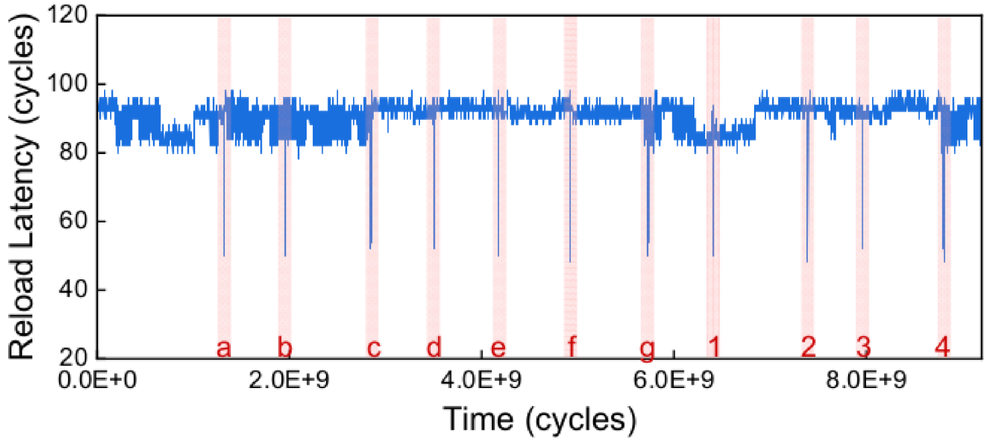


Fig. 8: The access latencies measured in Step 3 of Prefetch+Reload when a user types "abcdefg1234" in gedit; we monitor address 0x7b980 of libgdk.so.[7]

Prefetch-Based Channels in Transient Execution Attacks

Higher bandwidth

More leakage in the transient window

Conclusion

Observations on PREFETCHW: works on read-only data; execution time depends on the coherence state of the target data.

Covert channel attacks using PREFETCHW have very high capacities. Side channels attacks leak information from real-world applications.