AfterImage: Leaking Control Flow Data and Tracking Load Operations via Hardware Prefetcher

Yun Chen, Lingfeng Pei*, and Trevor E. Carlson*

**co-first author*

Introduction to Microarchitecture Attacks

- Microarchitecture Attacks:
  - Many rely on cache primitives and speculative execution.
  - Lack of study on not speculative-execution path.
- Our focus is on the ***prefetcher***
  - Bringing data into cache in advance to improve performance.
  - Located in the processor ***back-end*** and ***not speculative-execution dependent***.

Outline

- Reverse-Engineering Intel IP-Stride Prefetcher
- Threat Model and Experimental Setup
- AfterImage Attack Flow
- Breaking Different Levels of Isolation
- Attacking Real-World Application via AfterImage
- A Lightweight Defense
- Conclusion

Reverse-Engineering Intel IP-Stride Prefetcher

- IP-stride prefetcher:
  - Tracks the Instruction Pointer (IP) of the load, e.g., 0x40285c
  - Records the strided access pattern, e.g., 2
  - Predicts the memory access address and loads it in advance

```
for (int i = 0; i < 100; i++)
    0x40285c: load array[i * 2]
```

NUS National University of Singapore

Reverse-Engineering Intel IP-Stride Prefetcher

- Index policy of IP-stride prefetcher in Intel:
  - 24 entries
  - **Indexed by lower 8 bits of IP**
  - **No extra tag (e.g., PID, TID) checking**
  - A potential contention resource!

```
for (int i = 0; i < 100; i++)
    0x....5c: load array[i * 2]
```

Reverse-Engineering Intel IP-Stride Prefetcher

- Stride Update Policy of IP-stride prefetcher in Intel:

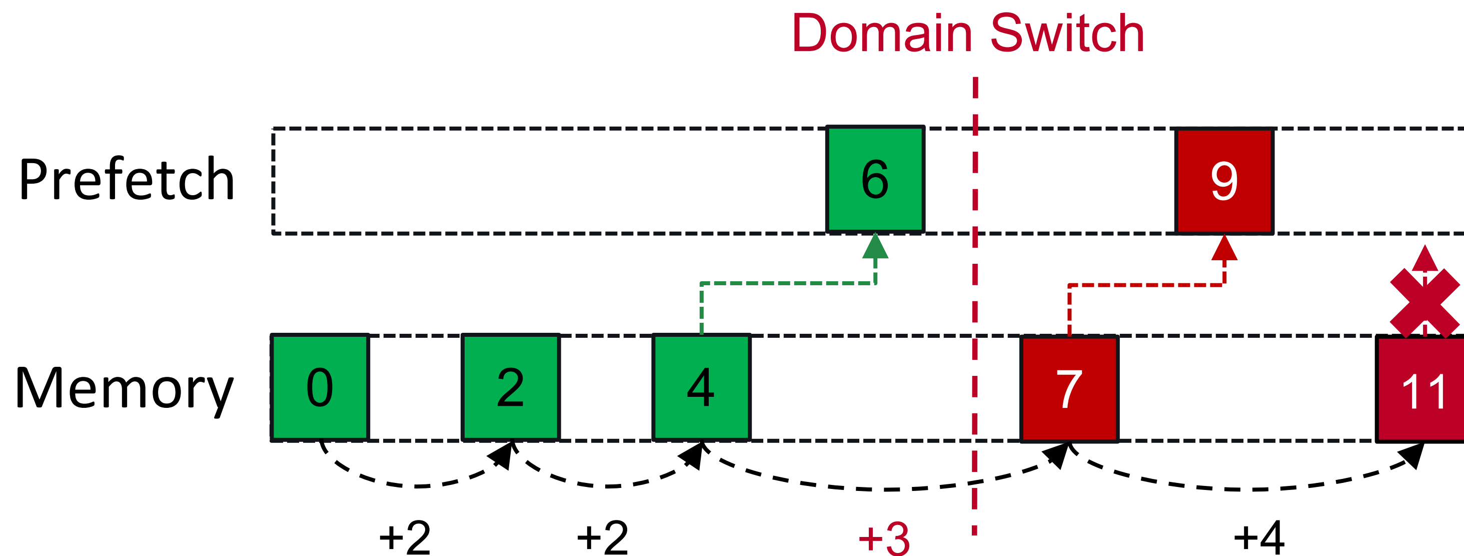
```
for (int i = 0; i < 100; i++)
    0x....5c: load array[i * 2]
```

Reverse-Engineering Intel IP-Stride Prefetcher

- Stride Update Policy of IP-stride prefetcher in Intel:
  - Enable prefetching if the confidence reaches **2**.
  - First prefetch then update. Make a potential attack channel.

```
for (int i = 0; i < 100; i++)
    0x....5c: load array[i * 2]
```

IP	Last Addr	Stride	Conf.
5c	array[11]	4	1
	.		
	.		
	.		

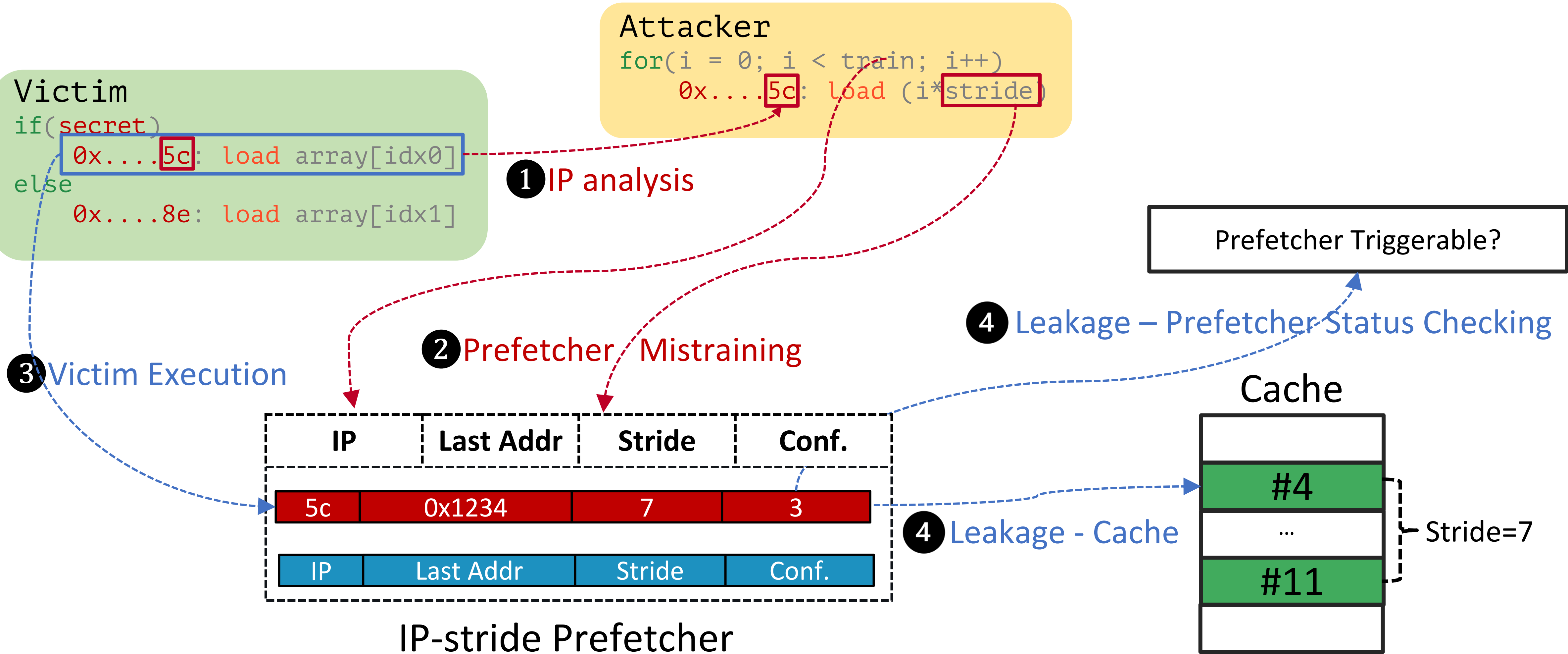


Threat Model and Experimental Setup

- We assume the attacker can analyze the victim's binary.
- We assume the attacker is running on the same physical core with the victim.

Experiment Machines	i7-4770 (Haswell) i7-9700 (Coffee Lake)
OS	Ubuntu 18.04
Kernel Version	5.4.0
(K)ASLR	Enable
Compiler	GCC 8.4.0 with –O0
DRAM	DDR4 2 x 8G, 1330.1 MHz

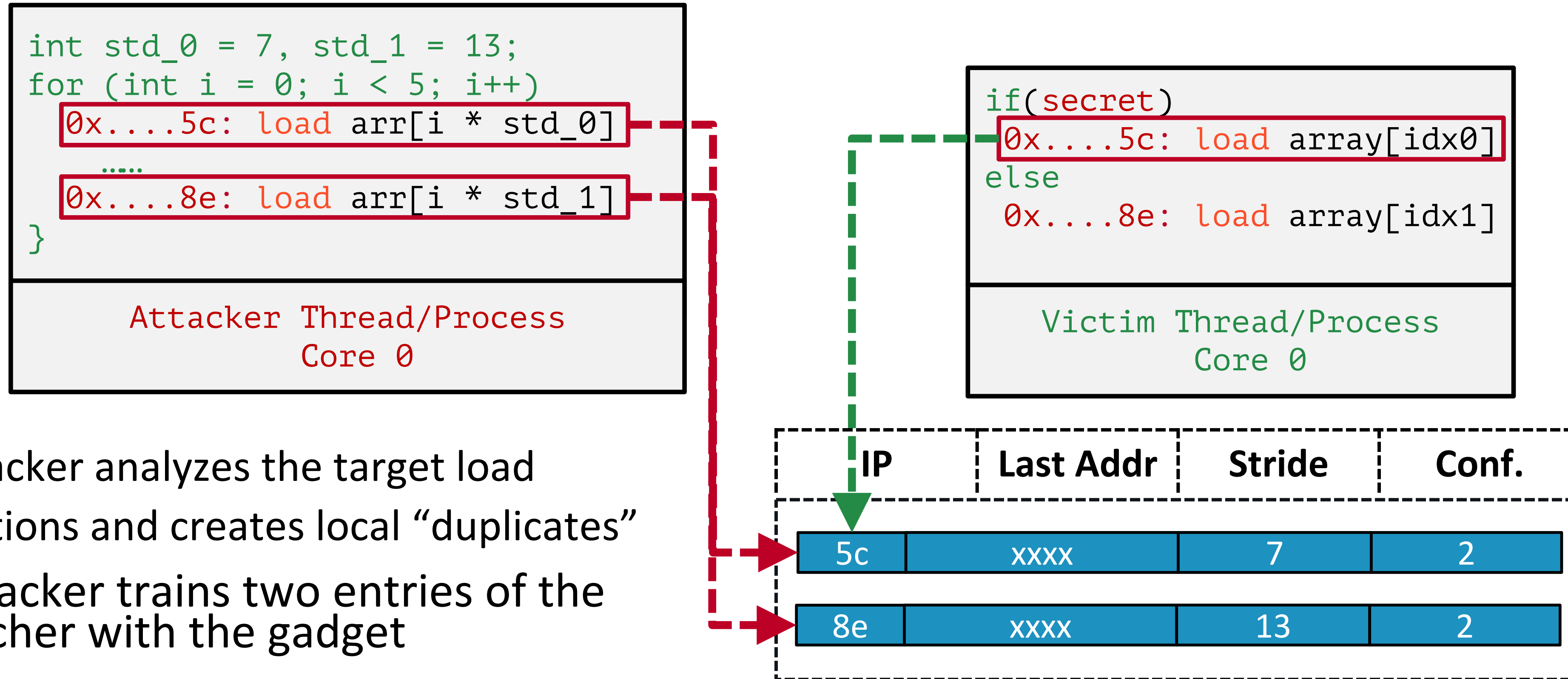
AfterImage Attack Flow



Outline

- Reverse-Engineering Intel IP-Stride Prefetcher
- Threat Model and Experimental Setup
- AfterImage Attack Flow
- Breaking Different Levels of Isolation
- Attacking Real-World Application via AfterImage
- A Lightweight Defense
- Conclusion

Breaking User-User Isolation



❶ Attacker analyzes the target load instructions and creates local "duplicates"

❷ Attacker trains two entries of the prefetcher with the gadget

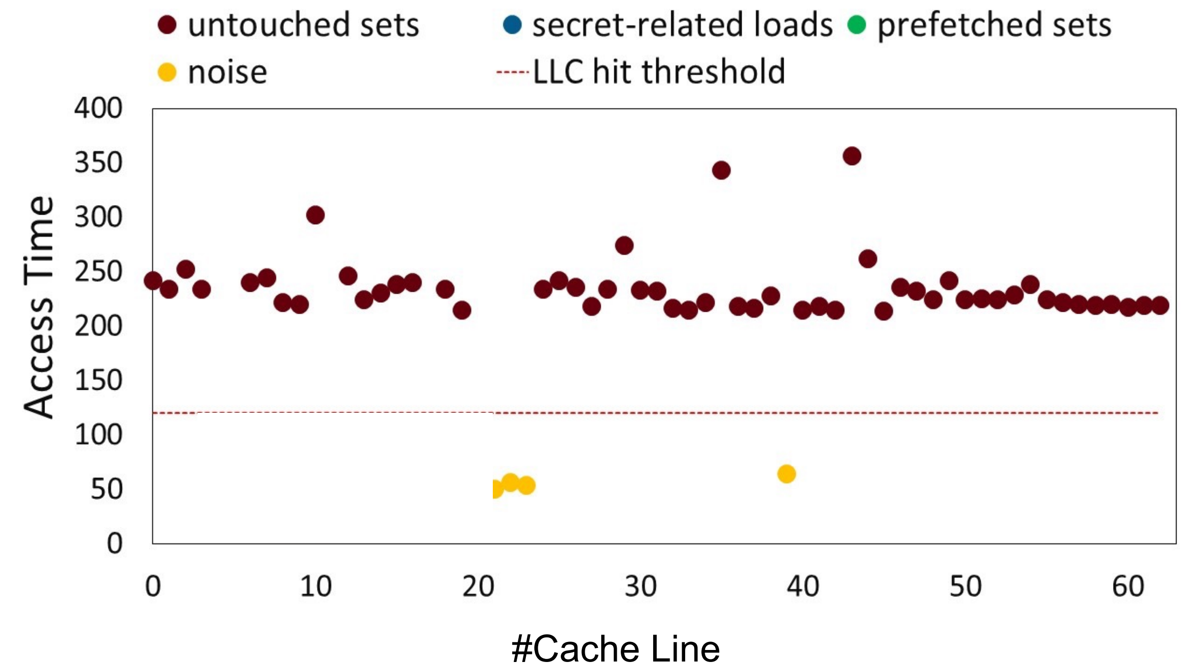
❸ Victim executes the target branch.

❹ Attacker detects the existence of **stride** in cache.

Breaking User-User Isolation Results

Variant 1: cross processes

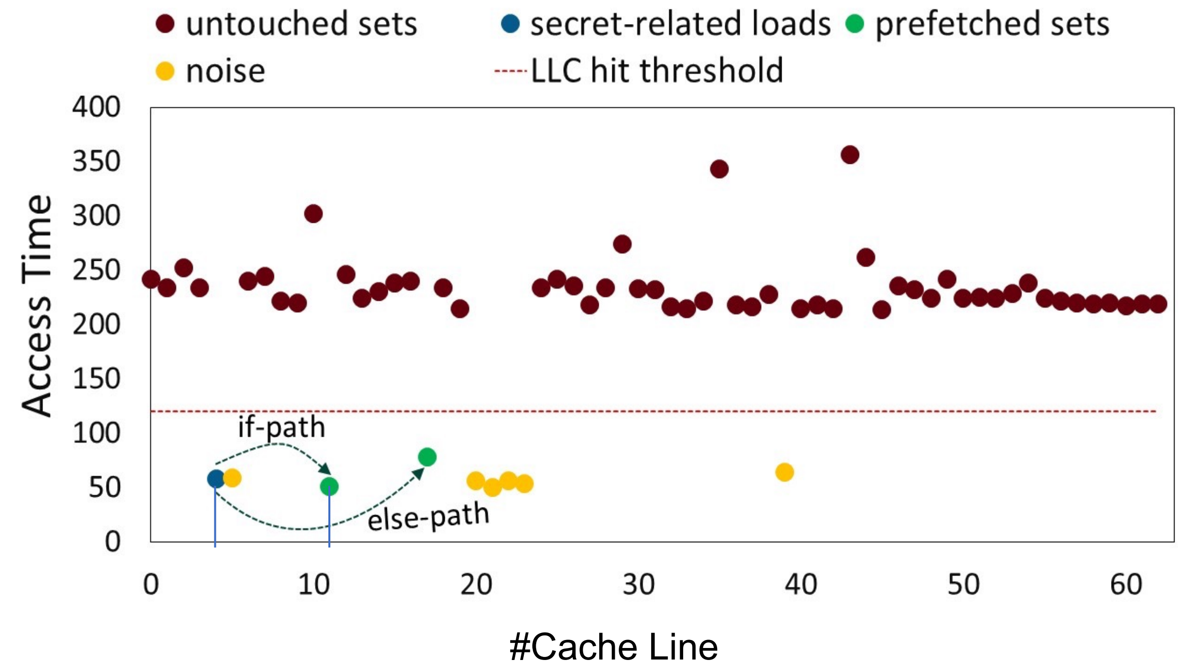
- Cache primitive:
  - Modified Flush+Reload
- Stride:
  1. If-path: 7 cache lines stride
  2. Else-path: 13 cache lines stride



Breaking User-User Isolation Results

Variant 1: cross processes

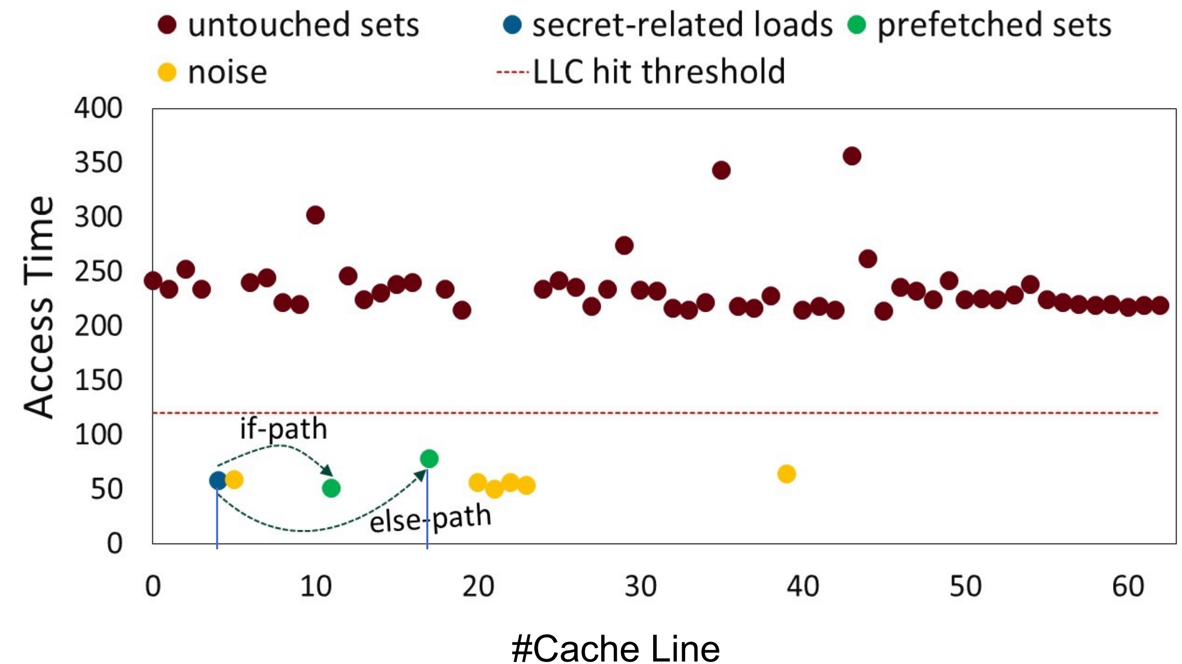
- Cache primitive:
  - Modified Flush+Reload
- Stride:
  1. If-path: 7 cache lines stride
  2. Else-path: 13 cache lines stride



Breaking User-User Isolation Results

Variant 1: cross processes

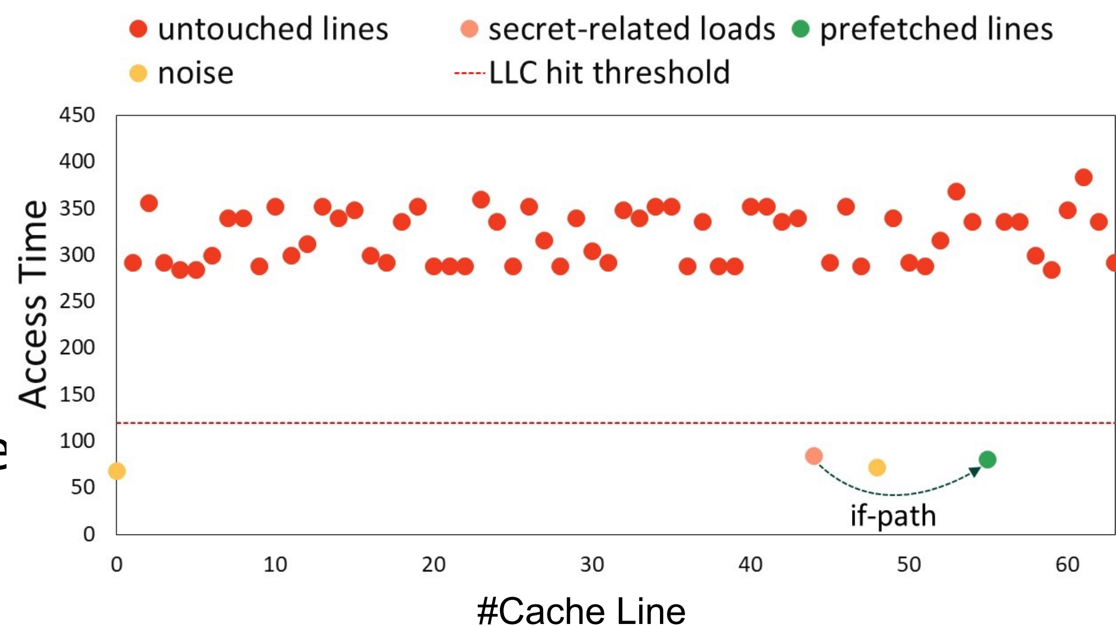
- Cache primitive:
  - Modified Flush+Reload
- Stride:
  1. If-path: 7 cache lines stride
  2. Else-path: 13 cache lines stride
- Result:
  - 7 exists (11 – 4)
  - 13 exists (17 – 4)



Breaking User-Kernel/SGX Isolation Results

Variant 2: cross user-kernel/SGX isolation

- Cache primitive:
  - Modified Flush+Reload
- Stride:
  1. If-path: 13 cache lines stride
- Result:
  - 13 exists (56 – 43)



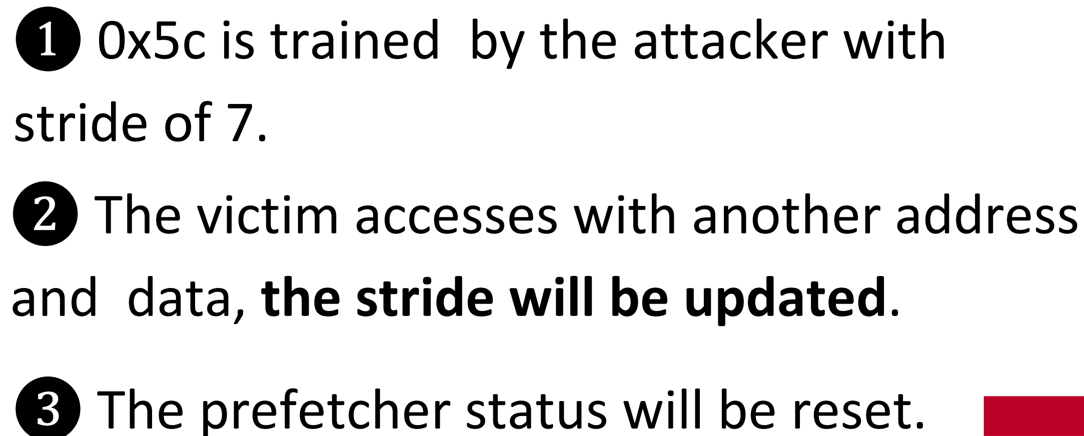
Attack Real-World Application via AfterImage

```
int std_0 = 7;
for (int i = 0; i < 5; i++)
  0x....5c: load arr[i * std_0]
}
```

Attacker Thread/Process
Core 0

```
if(secret)
  0x....5c: load arr1[idx0]
else
  0x....8e: load arr1[idx1]
```

Victim Thread/Process
Core 0



Prefetcher Status Checking (PSC) Technique

❶ 0x5c is trained by the attacker with stride of 7.

❷ The victim accesses with another address and data, **the stride will be updated**.

❸ The prefetcher status will be reset.

IP	Last Addr	Stride	Conf.
5c	arr1[x]	N	1

```
if(secret) ——→ 1
  0x....5c: load arr1[idx0]
```

Victim's Branch Executed

Attack Real-World Application via AfterImage

- Montgomery-Ladder RSA[1,2]
  - Why it is timing-constant:
    - Different directions always execute the same function call.
    - Only inputs are different.
  - Private key determines the branch direction.

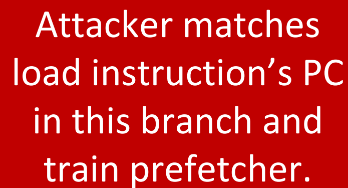
```
1   for(i=0; i<len(key); i++)
2   {
3       if(key[i] & 1)
4           ...
5           multiply_add();
6           clflush();
7       else
8           ...
9           multiply_add();
10          clflush();
11  }
```

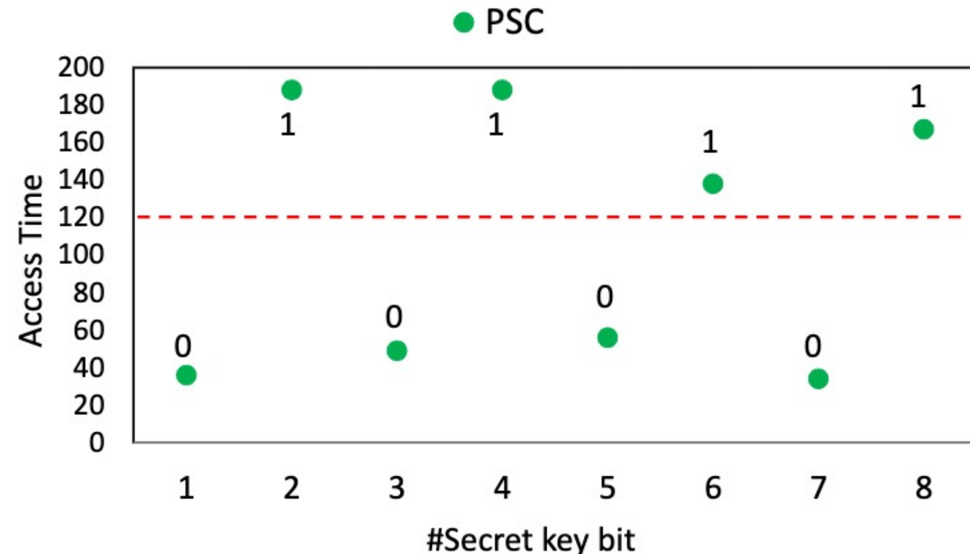
[1] https://github.com/merinjo/RSA-Montgomery-Ladder-Implementation
[2] OpenSSL 1.0.1e

Attack Real-World Application via AfterImage

- We break timing-constant RSA within 188 mins
  - Some distinguished load instructions are generated in different directions.
  - We leverage PSC technique to avoid using cache primitives.

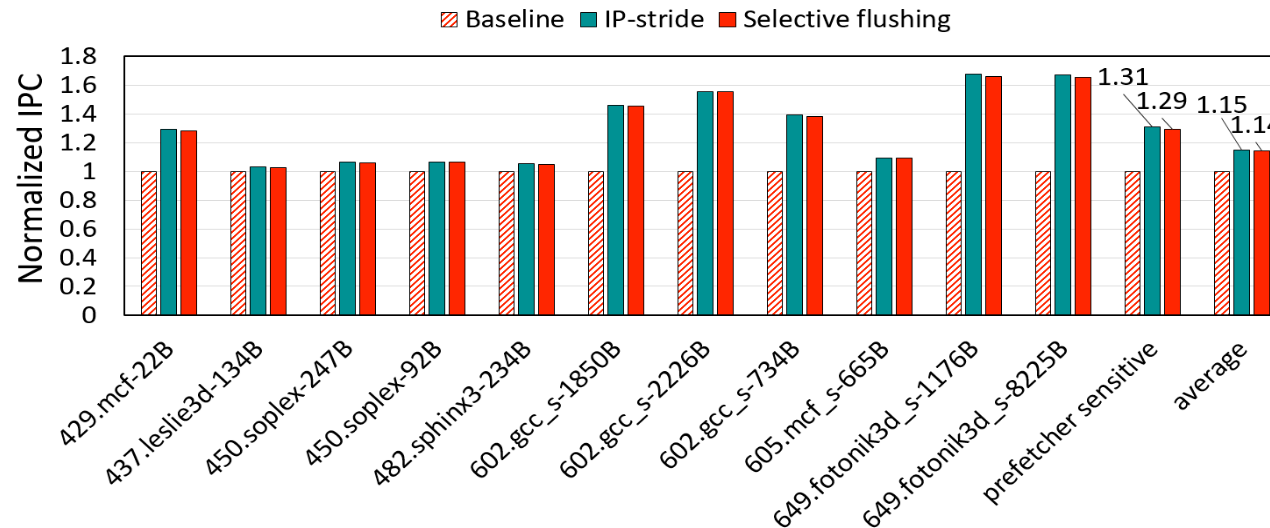
```
1   for(i=0; i<len(key); i++)
2   {
3       if(key[i] & 1)
4           ...
5           multiply_add();
6           clflush();
7       else
8           ...
9           multiply_add();
10          clflush();
11  }
```





Lightweight Defense

- Defense: **Clear the prefetcher at the context switch**
- Implementation: ChampSim
- Overhead: less than 0.2%, disable prefetcher introduce 15% overhead.



Conclusion

1. We reverse-engineer Intel IP-stride prefetcher.
2. We leak control flow data and track load instruction's timing information across different privilege regions.
3. We extract the private key of the timing-constant RSA.
4. We propose a defense with 0.2% perf. overhead.

AfterImage: Leaking Control Flow Data and Tracking Load Operations via Hardware Prefetcher

Yun Chen, Lingfeng Pei*, and Trevor E. Carlson*

**co-first author*