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

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

```c
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:

```c
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.

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

<table>
<thead>
<tr>
<th>IP</th>
<th>Last Addr</th>
<th>Stride</th>
<th>Conf.</th>
</tr>
</thead>
<tbody>
<tr>
<td>5c</td>
<td>array[11]</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

- Domain Switch

- Memory

- Prefetch
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) |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>Ubuntu 18.04</td>
</tr>
<tr>
<td>Kernel Version</td>
<td>5.4.0</td>
</tr>
<tr>
<td>(K)ASLR</td>
<td>Enable</td>
</tr>
<tr>
<td>Compiler</td>
<td>GCC 8.4.0 with –O0</td>
</tr>
<tr>
<td>DRAM</td>
<td>DDR4 2 x 8G, 1330.1 MHz</td>
</tr>
</tbody>
</table>
AfterImage Attack Flow

1. **IP analysis**
   - Attacker:
     ```c
     for(i = 0; i < train; i++)
     0x....5c: load (i*stride)
     ```

2. **Prefetcher Mistraining**
   - Victim Execution:
     ```c
     if(secret)
     0x....5c: load array[idx0]
     else
     0x....8e: load array[idx1]
     ```

3. **Victim Execution**

4. **Leakage - Prefetcher Status Checking**
   - Prefetcher Triggerable?

<table>
<thead>
<tr>
<th>IP</th>
<th>Last Addr</th>
<th>Stride</th>
<th>Conf.</th>
</tr>
</thead>
<tbody>
<tr>
<td>5c</td>
<td>0x1234</td>
<td>7</td>
<td>3</td>
</tr>
</tbody>
</table>

**IP-stride Prefetcher**

- Stride=7
- Leakage - Cache

**Cache**
- #4
- #11
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

1. Attacker analyzes the target load instructions and creates local “duplicates”
2. Attacker trains two entries of the prefetcher with the gadget
3. Victim executes the target branch.
4. 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

Prefetcher Status Checking (PSC) Technique

1. 0x5c is trained by the attacker with stride of 7.
2. The victim accesses with another address and data, the stride will be updated.
3. The prefetcher status will be reset.
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.

```c
for(i=0; i<len(key); i++)
{
    if(key[i] & 1)
    {
        ... multiply_add();
        clflush();
    }
    else
    {
        ... multiply_add();
        clflush();
    }
}
```

[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.

```c
for(i=0; i<len(key); i++)
{
    if(key[i] & 1)
        ...
        multiply_add();
        clflush();
    else
        ...
        multiply_add();
        clflush();
}
```

Attacker matches load instruction’s PC in this branch and train prefetcher.
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