Timing Channels

Principles

Information leakage through timing of events
- Typically by observing response latencies or own execution speed
- Covert channel: Information flow that bypasses the security policy
- Side channel: Covert channel exploitable without insider help

Causes of Timing Channels

Algorithmic
- If (secret) {
  short_operation(...);
} else {
  long_operation(...);
}

Resource Contention
- Software resources
- OS abstractions
- Buffer cache...
- Hardware resources
- Caches etc
- Not visible at ISA (HW-SW contract)

OS problem or not?

Shared Hardware: Stateless Interconnect

H/W is bandwidth-limited
- Interference during concurrent access
- Generally reveals no data or addresses
- Must encode info into access patterns
- Only usable as covert channel, not side channel
Shared Hardware: Stateful Resources

- **HW is capacity-limited**
  - Interference during concurrent access
  - Time-shared access
  - Collisions reveal addresses

- **Usable as side channel**

Can be any state-holding microarchitectural feature:
- CPU caches
- Branch predictor
- Pre-fetcher state machines

Timing Channels

Example: LLC Side Channel

Methodology: Prime and Probe

1. Fill cache with own data
2. Touch cache lines
3. Traverse cache, measure execution time

Victim executes normally
Spy observes

Challenge: Slow LLC Access Times

- **L1 (32 KiB) probe:**
  - 64 sets * 8 ways * 4 cycles = 2,048 cycles
- **Small last-level cache (6 MiB):**
  - 8,192 sets * 12 ways * ~30 cycles = ~3,000,000 cycles

Approach:
- Probe one or a few cache sets at a time
- Find “interesting” sets (“eviction set”) by looking for patterns

Searching for square Code

Expected pattern: Bursts of activity separated by longer or shorter intervals indicating modular reduction operation

```
long_int modular_reduction(long_int r, long_int h, m, e) {
    long_int res = 1;
    for (i = n - 1; i >= 0; i--)
        if (e[i]) {
            res = mod (res * r, m);
        }
    return res;
}
```

**Expected pattern:** Bursts of activity separated by longer or shorter intervals indicating modular reduction operation

Can read out bits of exponent!

Example: Look for square code in square-and-multiply exponentiation of GnuPG

Searching for square Code

Expected pattern: Bursts of activity separated by longer or shorter intervals indicating modular reduction operation

Can read out bits of exponent!

**[Liu S&P’15]**

0 50 100 150 200

Cache set
Timing Channels
Evaluating Hardware

Timing-Channel Prevention: Partition HW
- Temporally partition
-Spatially partition
Flush
Need both
- Flushing useless with concurrent access:
  - HW threads (SMT)
  - Multicore
- Cannot spatially partition on-core caches (L1, TLB, branch predictor, pre-fetchers)
  - virtually-indexed
  - OS cannot control
- No partitioning support for interconnects!

Evaluating Intra-Core Channels
- Methodology:
  - Flush all caches on context switch
  - using all flush ops provided by HW
  - Run prime/probe covert channel attack

Methodology: Channel Matrix
- Horizontal variation indicates channel
- Raw I-cache channel
  - Intel Sandy Bridge
- Channel Matrix:
  - Conditional probability of observing time \( t \), given input \( n \).
  - Represented as heat map: bright = high probability

I-Cache Channel With Full State Flush
- No evidence of channel
- SMALL CHANNEL!

HiSilicon A53 Branch History Buffer
- Branch history buffer (BHB)
  - Prediction of branch taken
  - One-bit channel
  - All reset operations applied
Intel Haswell Branch Target Buffer

Branch target buffer
- Prediction of branch destination
- All reset operations applied

Result Summary: Measured Capacities

<table>
<thead>
<tr>
<th>Channel</th>
<th>Sandy Bridge</th>
<th>Haswell</th>
<th>Skylake</th>
<th>ARM A9</th>
<th>ARM A53</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 D-cache</td>
<td>raw flush</td>
<td>raw flush</td>
<td>raw flush</td>
<td>raw flush</td>
<td>raw flush</td>
</tr>
<tr>
<td>L1 I-cache</td>
<td>4.0 0.04</td>
<td>4.7 0.43</td>
<td>3.3 0.18</td>
<td>5.0 0.11</td>
<td>2.8 0.15</td>
</tr>
<tr>
<td>TLB</td>
<td>3.2 0.47</td>
<td>3.2 0.18</td>
<td>2.5 0.11</td>
<td>0.33 0.16</td>
<td>3.4 0.14</td>
</tr>
<tr>
<td>BTB</td>
<td>2.0 1.7</td>
<td>4.1 1.6</td>
<td>1.8 1.9</td>
<td>1.1 0.7</td>
<td>1.3 0.64</td>
</tr>
<tr>
<td>BHB</td>
<td>1.0 1.0</td>
<td>1.0 1.0</td>
<td>1.0 1.0</td>
<td>1.0 0.01</td>
<td>1.0 0.5</td>
</tr>
</tbody>
</table>

Intel Spectre Defences

- Intel added indirect branch control (IBC) feature, which closes most channels, but...

Speculating Desaster

Instruction Pipelining

- Nominally, the processor executes instructions one after the other
- Instruction execution consists of multiple steps
  - Each uses a different unit

Instruction Pipelining

- Nominally, the processor executes instructions sequentially
- Instruction execution consists of multiple steps
  - Each uses a different unit
- Pipelining concurrently instruction execution

Problem: Dependencies

- Instruction fetch
- Instruction decode
- Argument fetch
- Execute
- Write back
Out-of-Order Execution

- Execute instructions when data is available

Out-of-order is speculative!

Completed instructions wait in reorder buffer until all previous ones retired

Out-of-Order Execution

- Abandon instructions if never executed in program order

Also useful for branches

Speculative Execution and Branches

When execution reaches a branch:
- Predict outcome of branch
- Proceed (speculatively) along predicted branch

Correct prediction: All good

Mis-prediction: Abandon and resume

Minor problem: Speculation pollutes cache!

Speculating Desaster

Meltdown:
Speculative Load

Prefetch concurrent with permission check, load aborted

Meltdown: Speculative Loads

Prefetch concurrent with permission check, load aborted
Meltdown: Cache-Channel to Read

```
i = *pointer; y = array[i*256];
```

Found!!!

```
array array2 secret array_len
```

Cache

Meltdown: Full Kernel Memory Disclosure

- **User mode**
  - Kernel mode
  - Fix: Separate address spaces

- **User mode**
  - Kernel mode

- **RAM**
  - Does kernel hold secrets?

- **Physical memory map**

Spectre: Branch Prediction (Variant 1)

```
if (x < array_len) {
i = array[x];
y = array2[i*256];
}
```

Attacker

```
x < array_len
```

Branch not taken!

```
x
```

Cache

Attacker

```
x < array_len
```

Branch not taken!

```
x
```

Cache

Mispredict!

```
x < array_len
```

Cache polluted!

```
x array secret array_len
```

Attacker

```
x
```

Physical memory map

```
x
```

Cache

```
x < array_len
```

Attacker

```
x
```

Cache

```
x
```

Physical memory map

Reactions

Steve Smith, Corporate vice president, Intel:

"The processor is, in fact, operating as it is designed," Smith said. "And in every case, it's been this side-channel approach that the researchers used to gain information even while the processor is executing normally its intended functions."

Inevitable conclusion:

- This ISA is an insufficient contract for building secure systems
- We need a new hardware-software contract!