Evolution of micro-architectural attacks

Clémentine Maurice, CNRS, CRIStAL
17 December 2021—32nd HP/HPE (Virtual) Colloquium On Information Security
Attacks on micro-architecture

- **hardware** usually modeled as an abstract layer behaving correctly
• **hardware** usually modeled as an abstract layer behaving correctly, but possible attacks
• **hardware** usually modeled as an abstract layer behaving correctly, but possible attacks
  - faults: bypassing software protections by causing **hardware errors**
  - side channels: observing **side effects** of hardware on computations
Attacks on micro-architecture

- **hardware** usually modeled as an abstract layer behaving correctly, but possible attacks
  - faults: bypassing software protections by causing **hardware errors**
  - side channels: observing **side effects** of hardware on computations

**identification**

[Graph showing cache hits and cache misses over access time in CPU cycles]
Attacks on micro-architecture

- **hardware** usually modeled as an abstract layer behaving correctly, but possible attacks
  - faults: bypassing software protections by causing **hardware errors**
  - side channels: observing **side effects** of hardware on computations

**identification**

<table>
<thead>
<tr>
<th>Access time [CPU cycles]</th>
<th>Number of accesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>$10^1$</td>
</tr>
<tr>
<td>200</td>
<td>$10^2$</td>
</tr>
<tr>
<td>300</td>
<td>$10^3$</td>
</tr>
<tr>
<td>400</td>
<td>$10^4$</td>
</tr>
</tbody>
</table>

**attack**

- retrieving secret keys, keystroke timings
- bypassing OS security (ASLR)
Attacker model

Hardware-based attacks
a.k.a. physical attacks

Physical access to hardware
→ embedded devices

Software-based attacks
a.k.a. micro-architectural attacks

Co-located or remote attacker
→ complex systems
Side-channel attacks
From small optimizations...

- new microarchitectures yearly

- 2011: Sandy Bridge
- 2012: Ivy Bridge
- 2013: Haswell
- 2014: Broadwell
- 2015: Skylake
- 2016: Kaby Lake
- 2017: Coffee Lake
- 2018: Whiskey Lake
- 2019: Comet Lake/Ice Lake
- 2020: Tiger Lake
From small optimizations...

- new microarchitectures yearly
- performance improvement $\approx 5\%$
From small optimizations...

- new microarchitectures yearly
- performance improvement $\approx 5\%$
- very small optimizations: caches, branch prediction...
To microarchitectural side-channel attacks

- microarchitectural side channels come from these optimizations
  - several processes are sharing microarchitectural components
  - attacker infers information from a (vulnerable) victim process via hardware usage
  - pure-software attacks by unprivileged processes
    - sequences of benign-looking actions
      → hard to detect
To microarchitectural side-channel attacks

- microarchitectural side channels come from these optimizations
- several processes are sharing microarchitectural components
... To microarchitectural side-channel attacks

- microarchitectural side channels come from these optimizations
- several processes are sharing microarchitectural components
- attacker infers information from a (vulnerable) victim process via hardware usage
To microarchitectural side-channel attacks

- Microarchitectural side channels come from these optimizations.
- Several processes are sharing microarchitectural components.
- Attacker infers information from a (vulnerable) victim process via hardware usage.
- Pure-software attacks by unprivileged processes.
... To microarchitectural side-channel attacks

- microarchitectural side channels come from these optimizations
- several processes are sharing microarchitectural components
- attacker infers information from a (vulnerable) victim process via hardware usage
- pure-software attacks by unprivileged processes
- sequences of benign-looking actions → hard to detect
Historical recap of past attacks
Historical recap of past attacks

Recent advances
Outline

Historical recap of past attacks

Recent advances

Future and challenges
Historical Recap
Micro-architectural attacks: Two faces of the same coin

Implementation

Algorithm 1: Square-and-multiply exponentiation

Input: base $b$, exponent $e$, modulus $n$
Output: $b^e \mod n$

$X \leftarrow 1$
for $i \leftarrow \text{bitlen}(e)$ downto 0 do
  $X \leftarrow \text{multiply}(X, X)$
  if $e_i = 1$ then
    $X \leftarrow \text{multiply}(X, b)$
  end
end
return $X$

Hardware
Research questions

1. Which **software implementation** is vulnerable?

2. Which **hardware component** is vulnerable?
1. Which software implementation is vulnerable?

State of the art (more or less)

1. Spend too much time reading OpenSSL code
2. Find vulnerability
3. Exploit it manually using known side channel → e.g. CPU cache
4. Publish
5. goto step 1

2. Which hardware component leaks information?

State of the art (more or less)

1. Spend too much time reading Intel manuals
2. Find weird behavior in corner cases
3. Exploit it
4. Publish
5. goto step 1
From theoretical to practical cache attacks

• first theoretical attack in 1996 by Kocher
• first practical attack on RSA in 2005 by Percival, on AES in 2006 by Osvik et al.
• renewed interest for the field in 2014 after Flush+Reload by Yarom and Falkner

---

Hyper-threading: Same-core attacks

• threads sharing one core share resources: L1, L2 cache, branch predictor
Possible side channels using components shared by a core?
Possible side channels using components shared by a core?

Stop sharing a core!
Caches on Intel CPUs

- L1 and L2 are private
- Last-level cache divided in slices
- Shared across cores
- Inclusive
Caches on Intel CPUs

- L1 and L2 are private
Caches on Intel CPUs

- L1 and L2 are private
- Last-level cache
  - Divided in slices
  - Shared across cores
  - Inclusive
Set-associative caches

<table>
<thead>
<tr>
<th>Address</th>
<th>0</th>
<th>16 17</th>
<th>25 26</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Index</td>
<td>Offset</td>
<td></td>
</tr>
<tr>
<td>Cache</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Set-associative caches

Data loaded in a specific set depending on its address
Data loaded in a specific set depending on its address

Several ways per set
Set-associative caches

Data loaded in a specific **set** depending on its address

Several **ways** per set

**Cache line** loaded in a specific way depending on the replacement policy
Cache attacks

- caches improve performance
• caches improve performance
• SRAM is expensive $\rightarrow$ small caches
Cache attacks

- caches improve performance
- SRAM is expensive $\rightarrow$ small caches
- different timings for memory accesses
Cache attacks

- caches improve performance
- SRAM is expensive → small caches
- different timings for memory accesses
  1. data is cached → cache hit → fast
Cache attacks

- Caches improve performance
- SRAM is expensive → small caches
- Different timings for memory accesses
  1. Data is cached → cache hit → fast
  2. Data is not cached → cache miss → slow
Cache attacks

- Caches improve performance
- SRAM is expensive → small caches
- Different timings for memory accesses
  1. Data is cached → cache hit → fast
  2. Data is not cached → cache miss → slow
- Cache attacks leverage this timing difference
Timing differences

![Bar chart showing cache hits and cache misses over access time in CPU cycles.](chart)

- **Access time [CPU cycles]**: 50, 100, 150, 200, 250, 300, 350, 400
- **Number of accesses**: 10^1, 10^2, 10^3, 10^4, 10^5, 10^6, 10^7
- **Cache hits** and **cache misses**
Cache attacks: Flush+Reload

Step 1: Attacker maps shared library (shared memory, in cache)
Cache attacks: Flush+Reload

Step 1: Attacker maps shared library (shared memory, in cache)
Cache attacks: Flush+Reload

**Step 1:** Attacker maps shared library (shared memory, in cache)

**Step 2:** Attacker flushes the shared cache line
Cache attacks: Flush+Reload

**Step 1:** Attacker maps shared library (shared memory, in cache)

**Step 2:** Attacker **flushes** the shared cache line

**Step 3:** Victim loads the data
Cache attacks: Flush+Reload

**Step 1:** Attacker maps shared library (shared memory, in cache)

**Step 2:** Attacker **flushes** the shared cache line

**Step 3:** Victim loads the data

**Step 4:** Attacker **reloads** the data
Flush+Reload: Applications

• cross-VM side channel attacks on crypto algorithms
  • RSA: 96.7% of secret key bits in a single signature
  • AES: full key recovery in 30000 dec. (a few seconds)
• covert channels in native environments cross-VM: 298 KBps

Flush+Reload: Pros and cons

- **high spatial resolution**: 1 cache line (64 Bytes)
Flush+Reload: Pros and cons

- high spatial resolution: 1 cache line (64 Bytes)
- but requires shared memory + clflush instruction
Flush+Reload: Pros and cons

- **high spatial resolution**: 1 cache line (64 Bytes)
- but requires shared memory + `clflush` instruction

→ memory deduplication between VMs
Possible side channels using memory deduplication?
Possible side channels using memory deduplication?

Disable memory deduplication!
Cache attacks: Prime+Probe

Victim address space

Cache

Attacker address space
Cache attacks: Prime+Probe

**Step 1:** Attacker primes, i.e., fills, the cache (no shared memory)
Cache attacks: Prime+Probe

Step 1: Attacker *primes*, *i.e.*, fills, the cache (no shared memory)

Step 2: Victim evicts cache lines while running
Cache attacks: Prime+Probe

**Step 1:** Attacker primes, *i.e.*, fills, the cache (no shared memory)

**Step 2:** Victim evicts cache lines while running
**Step 1**: Attacker primes, i.e., fills, the cache (no shared memory)

**Step 2**: Victim evicts cache lines while running

**Step 3**: Attacker probes data to determine if set has been accessed
Cache attacks: Prime+Probe

Step 1: Attacker primes, \textit{i.e.}, fills, the cache (no shared memory)

Step 2: Victim evicts cache lines while running

Step 3: Attacker probes data to determine if set has been accessed
Challenges with Prime+Probe

We need to evict caches lines without `clflush` or shared memory:

1. which addresses do we access to have congruent cache lines?
2. without any privilege?
3. and in which order do we access them?

We need:

1. an eviction set: addresses in the same set, in the same slice (issue #1 and #2)
2. an eviction strategy (issue #3)
Prime+Probe: Applications

• cross-VM side channel attacks on crypto algorithms:
  • El Gamal (sliding window): full key recovery in 12 min.
• tracking user behavior in the browser, in JavaScript
• covert channels between virtual machines in the cloud

C. Maurice et al. “Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud”. In: NDSS’17. 2017.
Possible side channels using components shared by a CPU?
Possible side channels using components shared by a CPU?

Stop sharing a CPU!?
Recent Advances
Increasing the attack surface
It’s not just caches: DRAM, GPU, TLB, CPU ports, Ring interconnect...!
It’s not just side channels: Fault attacks too!

Clockscrew: Exposing the Perils of Security-Oblivious Energy Management

Adrian Tang
Columbia University

Sudha Seshadri
Columbia University

Failing Bits in Memory Without Accessing Them:
An Experimental Study of DRAM Disturbance Errors

Yoongu Kim1 Ross Daly* Jeremy Kim1 Chris Fallin* Ji Hye Lee1 Donghyuk Lee1 Chris Wilkerson2 Konrad Lai Onur Mutlu1
1Carnegie Mellon University 2Intel Labs

Abstract. Memory isolation is a key property of a reliable and secure computing system — an access to one memory address should not have unintended side effects on data stored in other addresses. However, as DRAM process technology scales down to smaller dimensions, it becomes more difficult to prevent DRAM cells from electrically interacting with each other. In this paper, we examine the vulnerabilities of commodity disturbance errors. DRAM manufacturers have been employing a two-pronged approach: (i) improving inter-cell isolation through circuit-level techniques [22, 32, 49, 61, 73] and (ii) screening for disturbance errors during post-production testing [3, 4, 64]. We demonstrate that their efforts to contain disturbance errors have not always been successful, and that commercial DRAM chips have been slipping into the field.
Transient execution attacks
Transient execution attacks

- novel class of attacks $\neq$ side-channel attacks
- transient execution attacks leak the actual target data
- disclosed in 2018 with Spectre and Meltdown

https://transient.fail/
Transient execution attacks

- novel class of attacks $\neq$ side-channel attacks
- transient execution attacks leak the actual target data
- disclosed in 2018 with Spectre and Meltdown
- SO MANY VARIANTS

https://transient.fail/
Transient execution attacks

- CPU avoids waiting for input data or availability of execution units
  → out-of-order execution and speculation
- sequential semantics is preserved

• some instructions are never committed, i.e., finally executed
• instructions that cause an exception + following instructions
• instructions in branches that are mispredicted
  these instructions are called transient instructions
• architectural state → everything is fine
Transient execution attacks

- CPU avoids waiting for input data or availability of execution units
  → out-of-order execution and speculation
- sequential semantics is preserved
- some instructions are never committed, i.e., finally executed
  - instructions that cause an exception + following instructions
  - instructions in branches that are mispredicted
- these instructions are called transient instructions
Transient execution attacks

• CPU *avoids waiting* for input data or availability of execution units
  → out-of-order execution and speculation
• sequential semantics is preserved

• some instructions are never committed, *i.e.*, finally executed
  • instructions that cause an exception + following instructions
  • instructions in branches that are mispredicted
• these instructions are called **transient instructions**

• architectural state → everything is fine
Transient execution attacks

• attacker uses a covert channel to encode the secret
• issue: instructions not committed leave traces in microarchitecture
• microarchitectural state is not supposed to be visible...
• ... but we know how to recover the state of caches
Transient execution attacks

- attacker uses a **covert channel** to encode the secret
- issue: instructions not committed **leave traces in microarchitecture**
- microarchitectural state is not supposed to be visible...
- ... but we know how to **recover the state of caches**
- microarchitectural state → everything is **not fine**
Transient execution attacks

• attacker uses a **covert channel** to encode the secret
• issue: instructions not committed **leave traces in microarchitecture**
• microarchitectural state is not supposed to be visible...
• ... but we know how to **recover the state of caches**
• microarchitectural state → everything is **not fine**

• leaking kernel memory, recovering passwords...
Transient execution attacks

- attacker uses a **covert channel** to encode the secret
- issue: instructions not committed **leave traces in microarchitecture**
- microarchitectural state is not supposed to be visible...
- ... but we know how to **recover the state of caches**
- microarchitectural state $\rightarrow$ everything is **not fine**

- leaking kernel memory, recovering passwords...
- difficult to fix: lazy error handling was a bug, but speculative execution is a feature!
Recent advances

Porting micro-architectural attacks to the Web
Porting micro-architectural attacks to the Web

- side-channel attacks on the cache, DRAM, MMU, (...), and transient execution attacks like Spectre, ret2spec, RIDL, (...), are coming to web browsers
- very low-level attacks in a high-level language with many abstraction layers in between
- complex but not impossible to perform
- fundamentally hard or impossible to fix in the browser

JS and timers: A complicated history

JS and timers: A complicated history

• initial countermeasures: lowering timer resolution
• browsers are adopting better isolation between websites (e.g., Site Isolation) to counter transient execution attacks
• back to higher timer resolution for usability → side-channel attacks are possible again!

Automating vulnerability and side channel discovery
Automating vulnerability and side channel discovery

Vulnerability discovery

Static analysis

Dynamic analysis

CacheAudit USENIX Sec ’13
- ct-verif USENIX Sec ’16
- Binsec/Rel S&P ’20
- ... ...

CacheD USENIX Sec ’17
- MicroWalk ACSAC ’18
- DATA USENIX Sec ’18
- ABSynthe NDSS ’20
- ... ...
Future and Challenges
Challenges and questions

• lack of documentation on microarchitectural components
• which components are vulnerable to these attacks?
• which software is vulnerable to these attacks?
• why do we still manually find vulnerabilities when we have automated tools?
• how to prevent attacks based on performance optimizations without removing performance?

Conclusion

• first paper by Kocher in 1996: 25 years of research in this area
• domain still in expansion: increasing number of papers published since 2015
• adopted countermeasures mainly target cryptographic implementations
• still a lot more to discover!
• quick fixes don’t work
• still a lot more work needed to find satisfying countermeasures
Thank you!

Contact

✉️ clementine.maurice@inria.fr
/twitter @BloodyTangerine
Evolution of micro-architectural attacks

Clémentine Maurice, CNRS, CRIStAL
17 December 2021—32nd HP/HPE (Virtual) Colloquium On Information Security