Cache attacks:
Software side-channel and fault attacks

Clémentine Maurice, Graz University of Technology
May 11, 2017—Spring school on Security & Correctness in the IoT 2017
• Clémence Maurice
• PhD in computer science, Postdoc @ Graz University Of Technology
• @BloodyTangerine
• clementine.maurice@iaik.tugraz.at
Who am I

• Clémentine Maurice
• PhD in computer science, Postdoc @ Graz University Of Technology
• @BloodyTangerine
• clementine.maurice@iaik.tugraz.at

+ Secure Systems team: Daniel Gruss, Michael Schwarz, Moritz Lipp
Scope

• everyday hardware: servers, workstations, laptops, smartphones...
Scope

• everyday hardware: servers, workstations, laptops, smartphones...

• remote side-channel attacks
• safe software infrastructure → no bugs, e.g., Heartbleed
Side channels

- safe software infrastructure → no bugs, e.g., Heartbleed
- does not mean safe execution
Side channels

- **safe software** infrastructure $\rightarrow$ no bugs, e.g., Heartbleed
- does not mean safe execution
- information **leaks** because of the **hardware** it runs on
- no “bug” in the sense of a mistake $\rightarrow$ lots of performance optimizations
Side channels

- safe software infrastructure → no bugs, e.g., Heartbleed
- does not mean safe execution
- information leaks because of the hardware it runs on
- no “bug” in the sense of a mistake → lots of performance optimizations

→ crypto and sensitive info., e.g., keystrokes and mouse movements
Sources of leakage

- via power consumption, electromagnetic leaks
Sources of leakage

• via power consumption, electromagnetic leaks
• targeted attacks, physical access
• via shared hardware and microarchitecture
• remote attacks

Controller
Rikomagic MK802 IV

Loop antenna

MicroSD card

Antenna tuning capacitor

SDR receiver
FUNcube Dongle Pro+

Power
4xAA batteries

WiFi antenna

Pita bread
Sources of leakage

- via power consumption, electromagnetic leaks
  → targeted attacks, physical access
Sources of leakage

• via power consumption, electromagnetic leaks
  → targeted attacks, physical access
• via shared hardware and microarchitecture
• via power consumption, electromagnetic leaks
  → targeted attacks, physical access

• via shared hardware and microarchitecture
  → remote attacks
Shared hardware

- Memory
  - DRAM
  - Memory bus

- CPU
  - Branch prediction unit
  - Arithmetic logic unit
  - Data and instruction cache
From small optimizations to side channels

- new microarchitectures yearly

- 2008: Nehalem
- 2012: Sandy Bridge
- 2013: Ivy Bridge
- 2014: Haswell
- 2015: Broadwell
- 2016: Skylake

Very small optimizations: caches, branch prediction… leading to side channels. No documentation on this intellectual property.
From small optimizations to side channels

- New microarchitectures yearly
- Performance improvement $\approx 5\%$

Timeline:
- 2008: Nehalem
- 2012: Sandy Bridge
- 2013: Ivy Bridge
- 2014: Haswell
- 2015: Broadwell
- 2016: Skylake
From small optimizations to side channels

- new microarchitectures yearly
- performance improvement $\approx 5\%$
- very **small optimizations**: caches, branch prediction...
From small optimizations to side channels

- new microarchitectures yearly
- performance improvement \( \approx 5\% \)
- very small optimizations: caches, branch prediction...
- ... leading to side channels
From small optimizations to side channels

- New microarchitectures yearly
- Performance improvement $\approx 5\%$
- Very small optimizations: caches, branch prediction...
- ... leading to side channels
- No documentation on this intellectual property
Today’s CPU complexity

• “Intel x86 documentation has more pages than the 6502 has transistors”

Today’s CPU complexity

- “Intel x86 documentation has more pages than the 6502 has transistors”
- 6502: 8-bit microprocessor, used in the Apple II, Commodore 64, Atari 800...

Today’s CPU complexity

• “Intel x86 documentation has more pages than the 6502 has transistors”
• 6502: 8-bit microprocessor, used in the Apple II, Commodore 64, Atari 800…
  • year: 1975 → 3510 transistors

---

Today’s CPU complexity

- “Intel x86 documentation has more pages than the 6502 has transistors”
- 6502: 8-bit microprocessor, used in the Apple II, Commodore 64, Atari 800...
  - year: 1975 → 3510 transistors
- 22-core Intel Xeon Broadwell-E5

Today's CPU complexity

- “Intel x86 documentation has more pages than the 6502 has transistors”
- 6502: 8-bit microprocessor, used in the Apple II, Commodore 64, Atari 800...
  - year: 1975 → 3510 transistors
- 22-core Intel Xeon Broadwell-E5
  - year: 2016 → 7.2 billion transistors

Today’s CPU complexity

- “Intel x86 documentation has more pages than the 6502 has transistors”
- 6502: 8-bit microprocessor, used in the Apple II, Commodore 64, Atari 800...
  - year: 1975 → 3510 transistors
- 22-core Intel Xeon Broadwell-E5
  - year: 2016 → 7.2 billion transistors
- Intel Software Developer’s Manuals (sept. 2016): 4670 pages

Today’s CPU complexity

• “Intel x86 documentation has more pages than the 6502 has transistors”
  • 6502: 8-bit microprocessor, used in the Apple II, Commodore 64, Atari 800…
    • year: 1975 → 3510 transistors
  • 22-core Intel Xeon Broadwell-E5
    • year: 2016 → 7.2 billion transistors
• Intel Software Developer’s Manuals (sept. 2016): 4670 pages
• (there are actually more manuals than just the SDM)

• Background
• Cache covert channels
• Side-channel attacks on crypto, keystrokes, and KASLR
• Fault attacks on DRAM in JavaScript
• Countermeasures and conclusion
Cache Attack Techniques
# MOV—Move

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
<th>Op/En</th>
<th>64-Bit Mode</th>
<th>Compat/Leg Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>88 8r</td>
<td>MOV r/m8,r8</td>
<td>MR</td>
<td>Valid</td>
<td>Valid</td>
<td>Move r8 to r/m8.</td>
</tr>
<tr>
<td>REX + 8B 8r</td>
<td>MOV r/m8***,r8***</td>
<td>MR</td>
<td>Valid</td>
<td>N.E.</td>
<td>Move r8 to r/m8.</td>
</tr>
<tr>
<td>89 8r</td>
<td>MOV r/m16,r16</td>
<td>MR</td>
<td>Valid</td>
<td>Valid</td>
<td>Move r16 to r/m16.</td>
</tr>
<tr>
<td>89 8r</td>
<td>MOV r/m32,r32</td>
<td>MR</td>
<td>Valid</td>
<td>Valid</td>
<td>Move r32 to r/m32.</td>
</tr>
<tr>
<td>REX,W + 89 8r</td>
<td>MOV r/m64,r64</td>
<td>MR</td>
<td>Valid</td>
<td>N.E.</td>
<td>Move r64 to r/m64.</td>
</tr>
<tr>
<td>8A 8r</td>
<td>MOV r8,r/m8</td>
<td>RM</td>
<td>Valid</td>
<td>Valid</td>
<td>Move r/m8 to r8.</td>
</tr>
<tr>
<td>REX + 8A 8r</td>
<td>MOV r8***,r/m8***</td>
<td>RM</td>
<td>Valid</td>
<td>N.E.</td>
<td>Move r/m8 to r8.</td>
</tr>
<tr>
<td>8B 8r</td>
<td>MOV r16,r/m16</td>
<td>RM</td>
<td>Valid</td>
<td>Valid</td>
<td>Move r/m16 to r16.</td>
</tr>
<tr>
<td>8B 8r</td>
<td>MOV r32,r/m32</td>
<td>RM</td>
<td>Valid</td>
<td>Valid</td>
<td>Move r/m32 to r32.</td>
</tr>
<tr>
<td>REX,W + 8B 8r</td>
<td>MOV r/m64,r64</td>
<td>RM</td>
<td>Valid</td>
<td>N.E.</td>
<td>Move r/m64 to r64.</td>
</tr>
<tr>
<td>8C 8r</td>
<td>MOV r/m16,Sreg**</td>
<td>MR</td>
<td>Valid</td>
<td>Move segment register to r/m16.</td>
<td></td>
</tr>
<tr>
<td>REX,W + 8C 8r</td>
<td>MOV r/m64,Sreg**</td>
<td>MR</td>
<td>Valid</td>
<td>Move zero extended 16-bit segment register to r/m64.</td>
<td></td>
</tr>
<tr>
<td>8E 8r</td>
<td>MOV Sreg,r/m16**</td>
<td>RM</td>
<td>Valid</td>
<td>Move r/m16 to segment register.</td>
<td></td>
</tr>
<tr>
<td>REX,W + 8E 8r</td>
<td>MOV Sreg,r/m64**</td>
<td>RM</td>
<td>Valid</td>
<td>Move lower 16 bits of r/m64 to segment register.</td>
<td></td>
</tr>
<tr>
<td>A0</td>
<td>MOV AL,moffs8*</td>
<td>FD</td>
<td>Valid</td>
<td>N.E.</td>
<td>Move byte at (seg:offset) to AL.</td>
</tr>
<tr>
<td>REX,W + A0</td>
<td>MOV AL,moffs8*</td>
<td>FD</td>
<td>Valid</td>
<td>N.E.</td>
<td>Move byte at (offset) to AL.</td>
</tr>
<tr>
<td>A1</td>
<td>MOV AX,moffs16*</td>
<td>FD</td>
<td>Valid</td>
<td>Move word at (seg:offset) to AX.</td>
<td></td>
</tr>
<tr>
<td>A1</td>
<td>MOV EAX,moffs32*</td>
<td>FD</td>
<td>Valid</td>
<td>Move doubleword at (seg:offset) to EAX.</td>
<td></td>
</tr>
<tr>
<td>REX,W + A1</td>
<td>MOV RAX,moffs64*</td>
<td>FD</td>
<td>Valid</td>
<td>N.E.</td>
<td>Move quadword at (offset) to RAX.</td>
</tr>
</tbody>
</table>
### 64-Bit Mode Exceptions

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#GP(0)</td>
<td>If the memory address is in a non-canonical form.</td>
</tr>
<tr>
<td></td>
<td>If an attempt is made to load SS register with NULL segment selector when CPL = 3.</td>
</tr>
<tr>
<td></td>
<td>If an attempt is made to load SS register with NULL segment selector when CPL &lt; 3 and CPL ≠ RPL.</td>
</tr>
<tr>
<td>#GP(selector)</td>
<td>If segment selector index is outside descriptor table limits.</td>
</tr>
<tr>
<td></td>
<td>If the memory access to the descriptor table is non-canonical.</td>
</tr>
<tr>
<td></td>
<td>If the SS register is being loaded and the segment selector's RPL and the segment descriptor’s DPL are not equal to the CPL.</td>
</tr>
<tr>
<td></td>
<td>If the SS register is being loaded and the segment pointed to is a nonwritable data segment.</td>
</tr>
<tr>
<td></td>
<td>If the DS, ES, FS, or GS register is being loaded and the segment pointed to is not a data or readable code segment.</td>
</tr>
<tr>
<td></td>
<td>If the DS, ES, FS, or GS register is being loaded and the segment pointed to is a data or nonconforming code segment, but both the RPL and the CPL are greater than the DPL.</td>
</tr>
<tr>
<td>#SS(0)</td>
<td>If the stack address is in a non-canonical form.</td>
</tr>
<tr>
<td>#SS(selector)</td>
<td>If the SS register is being loaded and the segment pointed to is marked not present.</td>
</tr>
<tr>
<td>#PF(fault-code)</td>
<td>If a page fault occurs.</td>
</tr>
<tr>
<td>#AC(0)</td>
<td>If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.</td>
</tr>
<tr>
<td>#UD</td>
<td>If attempt is made to load the CS register.</td>
</tr>
<tr>
<td></td>
<td>If the LOCK prefix is used.</td>
</tr>
</tbody>
</table>
mov—What could go wrong?

- lots of exceptions for mov
mov—What could go wrong?

• lots of exceptions for mov
• but accessing data loads it to the cache
mov—What could go wrong?

- lots of exceptions for mov
- but accessing data loads it to the cache
  → side effects on computations!
Memory hierarchy

- data can reside in

- CPU Registers
- L1 Cache
- L2 Cache
- Memory
- Disk storage
• data can reside in
  • CPU registers
Memory hierarchy

- data can reside in
  - CPU registers
  - different levels of the CPU cache
Memory hierarchy

- data can reside in
  - CPU registers
  - different levels of the CPU cache
  - main memory
Memory hierarchy

- data can reside in
  - CPU registers
  - different levels of the CPU cache
  - main memory
  - disk storage
Caches on Intel CPUs

- L1 and L2 are private

Diagram showing Intel CPU architecture with cores, L1, L2, LLC slices, and ring bus.
Caches on Intel CPUs

- L1 and L2 are private
- Last-level cache

Diagram:
- Core 0, Core 1, Core 2, Core 3
- L1 and L2 caches for each core
- LLC slices 0 to 3
- Ring bus for shared LLC access
Caches on Intel CPUs

- L1 and L2 are private
- Last-level cache
  - Divided in slices
Caches on Intel CPUs

- L1 and L2 are private
- Last-level cache
  - Divided in slices
  - Shared across cores
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>Index</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cache
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
Timing differences

Access time [CPU cycles]

Number of accesses

cache hits

cache misses
Accurate timings

- very short timings
- \texttt{rdtsc} instruction: cycle-accurate timestamps
Accurate timings

- very short timings
- \texttt{rdtsc} instruction: cycle-accurate timestamps

```cpp
[...]
rdtsc
function()
rdtsc
[...]
```
Accurate timings

- do you measure what you think you measure?
Accurate timings

- do you measure what you think you measure?
- out-of-order execution
Accurate timings

- do you measure what you think you measure?
- out-of-order execution → what is really executed

```
rdtsc
function()    rdtsc
[...]         rdtsc
rdtsc
function()    [...]
```
Accurate timings

- use pseudo-serializing instruction `rdtscp` (recent CPUs)
Accurate timings

- use pseudo-serializing instruction `rdtscp` (recent CPUs)
- and/or use serializing instructions like `cpuid`
Accurate timings

- use *pseudo-serializing* instruction `rdtscp` (recent CPUs)
- and/or use *serializing* instructions like `cpuid`
- and/or use *fences* like `mfence`
Accurate timings

- use **pseudo-serializing** instruction `rdtscp` (recent CPUs)
- and/or use **serializing** instructions like `cpuid`
- and/or use **fences** like `mfence`

Cache attacks

- cache attacks → exploit timing differences of memory accesses
Cache attacks

- cache attacks → exploit timing differences of memory accesses
- attacker monitors which lines are accessed, not the content
Cache attacks

- cache attacks → exploit timing differences of memory accesses
- attacker monitors which lines are accessed, not the content
- covert channel: two processes communicating with each other
  - not allowed to do so, e.g., across VMs
Cache attacks

- cache attacks → exploit timing differences of memory accesses
- attacker monitors which lines are accessed, not the content
- covert channel: two processes communicating with each other
  - not allowed to do so, e.g., across VMs
- side-channel attack: one malicious process spies on benign processes
  - e.g., steals crypto keys, spies on keystrokes
Cache attacks techniques

- two (main) techniques
  1. **Flush+Reload** (Gullasch et al., Osvik et al., Yarom et al.)
  2. **Prime+Probe** (Percival, Osvik et al., Liu et al.)
- exploitable on x86 and ARM

---

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
What if there is no shared memory?
Inclusive property

- Inclusive LLC: superset of L1 and L2

![Diagram showing the relationship between core 0, core 1, L1, L2, and LLC]
• inclusive LLC: superset of L1 and L2
Inclusive property

- **inclusive** LLC: superset of L1 and L2

Inclusion

- data evicted from the LLC is also evicted from L1 and L2
- a core can evict lines in the private L1 of another core
- inclusive LLC: superset of L1 and L2

- data evicted from the LLC is also evicted from L1 and L2
- a core can evict lines in the private L1 of another core
• inclusive LLC: superset of L1 and L2
• data evicted from the LLC is also evicted from L1 and L2
Inclusive property

- **inclusive LLC**: superset of L1 and L2
- Data evicted from the LLC is also evicted from L1 and L2
- A core can **evict lines** in the private L1 of another core
Cache attacks: Prime+Probe

Victim address space

Cache

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

We need to evict cache 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?
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?
Last-level cache addressing

physical address

<table>
<thead>
<tr>
<th>35</th>
<th>17</th>
<th>6</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag</td>
<td>set</td>
<td>offset</td>
<td></td>
</tr>
</tbody>
</table>

line

slice 0 slice 1 slice 2 slice 3

H

offset: 30
set tag: physical address 0 61735
line 2 slice 0
Last-level cache addressing

- last-level cache → as many slices as cores
- **undocumented** hash function that maps a physical address to a slice
- designed for performance

For $2^k$ slices:

physical address
30 bits

\[ \text{slice } (o_0, \ldots, o_{k-1}) \]
\[ k \text{ bits} \]
Prime+Probe on recent processors?

Undocumented function $\rightarrow$ impossible to target the same set in the same slice
Let’s reverse-engineer the last-level cache!
Reverse engineering method

1. find some way to map **one address to one slice**
Reverse engineering method

1. find some way to map one address to one slice
2. repeat for every address with a 64B stride
Reverse engineering method

1. find some way to map one address to one slice
2. repeat for every address with a 64B stride
3. infer a function out of it
How to map addresses to slices?

- with performance counters (Maurice et al., 2015)
- with a timing attack
  - using `clflush` (using Gruss et al., 2016)
  - using memory accesses (Yarom et al., 2015)

---


How to map addresses to slices?

- with **performance counters** (Maurice et al., 2015)
- with a timing attack
  - using `clflush` (using Gruss et al., 2016)
  - using memory accesses (Yarom et al., 2015)

---

Mapping with performance counters

- event UNC_CBO_CACHE_LOOKUP counts accesses to a slice
Mapping with performance counters

- event `UNC_CBO_CACHE_LOOKUP` counts accesses to a slice

```
<table>
<thead>
<tr>
<th>Slice</th>
<th>CBo 0</th>
<th>CBo 1</th>
<th>CBo 2</th>
<th>CBo 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

```
0x3a0071010

H

CBo 0

slice 0

UNC_CBO_CACHE_LOOKUP

33
Mapping with performance counters

- event **UNC_CBO_CACHE_LOOKUP** counts accesses to a slice

```
0x3a0071090

H

CBo 0
slice 0
1

CBo 1
slice 1
0

CBo 2
slice 2
1

CBo 3
slice 3
0
```

**UNC_CBO_CACHE_LOOKUP**
Mapping with performance counters

- event `UNC_CBO_CACHE_LOOKUP` counts accesses to a slice

```
0x3a00710d0
  H
  CBo 0
  slice 0
  1

CBo 1
slice 1
0

CBo 2
slice 2
1

CBo 3
slice 3
1
```

`UNC_CBO_CACHE_LOOKUP`
1. translate virtual to physical address with /proc/pid/pagemap
Mapping a physical address to a slice

1. translate virtual to physical address with /proc/pid/pagemap
2. set up monitoring session
1. translate virtual to physical address with `/proc/pid/pagemap`
2. set up monitoring session
3. repeat access to a single address
1. translate virtual to physical address with `/proc/pid/pagemap`
2. set up monitoring session
3. repeat access to a single address
   → `clflush` is already counted as an access
Mapping a physical address to a slice

1. translate virtual to physical address with /proc/pid/pagemap
2. set up monitoring session
3. repeat access to a single address
   → clflush is already counted as an access
4. read UNC_CBO_CACHE_LOOKUP event for each CBo
Mapping a physical address to a slice

1. translate virtual to physical address with /proc/pid/pagemap
2. set up monitoring session
3. repeat access to a single address
   → clflush is already counted as an access
4. read UNC_CBO_CACHE_LOOKUP event for each CBo
5. slice is the one that has the maximum lookup events
Mapping physical addresses to slices

Number of lookup events

0x3a0071010 0x3a0071050 0x3a0071090 0x3a00710d0

CBo 0  CBo 1  CBo 2  CBo 3
Two cases:

1. linear function ($2^n$ number of cores): XOR of address bits
Inferring the function

Two cases:

1. linear function \((2^n \text{ number of cores})\): XOR of address bits
   - solve equations with linear algebra

2. non-linear function (the rest): more complicated
Inferring the function

Two cases:

1. linear function ($2^n$ number of cores): XOR of address bits
   - solve equations with linear algebra
   - brute force
Two cases:

1. linear function ($2^n$ number of cores): XOR of address bits
   - solve equations with linear algebra
   - brute force

2. non-linear function (the rest): more complicated
Last-level cache linear functions

3 functions, depending on the number of cores

<table>
<thead>
<tr>
<th>Address bit</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Function valid for Sandy Bridge, Ivy Bridge, Haswell, Broadwell
#1. Cache Covert Channels
Covert channel

- malicious privacy gallery app

- malicious weather widget
  no permissions except accessing the Internet
Covert channel

- malicious privacy gallery app
  - no permissions except accessing your images
Covert channel

- malicious privacy gallery app
  - no permissions except accessing your images
- malicious weather widget
Covert channel

- malicious privacy gallery app
  - no permissions except accessing your images
- malicious weather widget
  - no permissions except accessing the Internet
Covert channel

- malicious privacy gallery app
  - no permissions except accessing your images
- malicious weather widget
  - no permissions except accessing the Internet
Covert channel

- malicious privacy gallery app
  - no permissions except accessing your images
- malicious weather widget
  - no permissions except accessing the Internet
Covert channel

- malicious privacy gallery app
  - no permissions except accessing your images
- malicious weather widget
  - no permissions except accessing the Internet
General principle

• sender transmits bits with cache hits and misses
• receiver monitors a cache line or a cache set to receive 1 bit
General principle

- sender transmits bits with **cache hits** and **misses**
- receiver **monitors** a cache line or a cache set to receive 1 bit
- transmitting on several lines or sets to send several bits
General principle

- sender transmits bits with **cache hits and misses**
- receiver **monitors** a cache line or a cache set to receive 1 bit
- transmitting on several lines or sets to send several bits
- Flush+Reload: using **cache line** from a shared library/executable
  - sender: accesses the line to send ’1’, stays idle to send ’0’
  - receiver: reloads the line, then flushes it
General principle

- Sender transmits bits with cache hits and misses.
- Receiver monitors a cache line or a cache set to receive 1 bit.
- Transmitting on several lines or sets to send several bits.
- Flush+Reload: using cache line from a shared library/executable.
  - Sender: accesses the line to send '1', stays idle to send '0'.
  - Receiver: reloads the line, then flushes it.
- Prime+Probe: agreeing on a cache set.
  - Sender: primes the set to send '1', stays idle to send '0'.
  - Receiver: probes the set.
• programs competing for
  • the CPU cache
  • scheduling

---

C. Maurice et al. “Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud”. In: NDSS’17. 2017.
Issues

- programs competing for
  - the CPU cache
  - scheduling
- literature: covert channels stop working with **noise** on the machine

---

C. Maurice et al. “Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud”. In: *NDSS’17*. 2017.
Issues

- programs competing for
  - the CPU cache
  - scheduling
- literature: covert channels stop working with *noise* on the machine
- solution? “Just use error-correcting codes”

C. Maurice et al. “Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud”. In: *NDSS’17*. 2017.
• programs competing for
  • the CPU cache
  • scheduling

• literature: covert channels stop working with noise on the machine
• solution? “Just use error-correcting codes”

• let’s investigate this!

C. Maurice et al. “Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud”. In: NDSS’17. 2017.
Why can’t we just use error correcting codes?

Sender: 1 0 0 1 1 0
Receiver: 1 0 0 1 1 0

(a) Transmission without errors
Why can’t we just use error correcting codes?

(a) Transmission without errors

Sender: 1 0 0 1 1 0
Receiver: 1 0 0 1 1 0

(b) Noise: substitution error

Sender: 1 0 0 1 1 0
Receiver: 1 1 0 1 1 0
Why can’t we just use error correcting codes?

(a) Transmission without errors

Sender: 1 0 0 1 1 0
Receiver: 1 0 0 1 1 0

(b) Noise: substitution error

Sender: 1 0 0 1 1 0
Receiver: 1 1 0 1 1 0

(c) Sender descheduled: insertions

Sender: 1 0 0 0 0 0 1 1 0
Receiver: 1 0 0 0 0 0 1 1 0
Why can’t we just use error correcting codes?

(a) Transmission without errors

Sender 1 0 0 1 1 0
Receiver 1 0 0 1 1 0

(b) Noise: substitution error

Sender 1 0 0 1 1 0
Receiver 1 1 0 1 1 0

(c) Sender descheduled: insertions

Sender 1 0 0 0 0 1 1 0
Receiver 1 0 0 0 0 1 1 0

(d) Receiver descheduled: deletions

Sender 1 0 0 1 1 0
Receiver 1 0 1 1 0
Our robust covert channel

- **physical layer:**
  - transmits words as a sequence of ‘0’s and ‘1’s
  - deals with synchronization errors

- **data-link layer:**
  - divides data to transmit into packets
  - corrects the remaining errors
Physical layer: Sending ‘0’s and ‘1’s

- sender and receiver agree on one set
Physical layer: Sending ‘0’s and ‘1’s

- sender and receiver agree on one set
- receiver probes the set continuously
Physical layer: Sending ‘0’s and ‘1’s

- sender and receiver agree on one set
- receiver probes the set continuously
- sender transmits ‘0’ doing nothing
  → lines of the receiver still in cache → fast access
Physical layer: Sending ‘0’s and ‘1’s

- sender and receiver agree on one set
- receiver probes the set continuously
- sender **transmits ’0’** doing nothing
  → lines of the receiver still in cache → **fast access**
- sender **transmits ’1’** accessing addresses in the set
  → evicts lines of the receiver → **slow access**
Eviction set generation

- need a set of addresses in the same cache set and same slice
Eviction set generation

- need a set of addresses in the same cache set and same slice
- problem: slice number depends on all bits of the physical address
Eviction set generation

- need a set of addresses in the same cache set and same slice
- problem: slice number depends on all bits of the physical address

- we can build a set of addresses in the same cache set and same slice
Eviction set generation

- need a set of addresses in the **same cache set and same slice**
- problem: slice number depends on all bits of the physical address

- we can build a set of addresses in the **same cache set and same slice**
- without knowing **which slice**
Jamming agreement

sender eviction sets

Cache Sets

receiver eviction sets
Jamming agreement

sender eviction sets

#1
#2
#3
#4

prime

Cache Sets

 receiver eviction sets

S S S S S S S S S
Jamming agreement

sender eviction sets

#1
#2
#3
#4

receiver eviction sets

Cache Sets

S S S S S S S S S
R R R R R R R R R

prime
Jamming agreement

sender eviction sets

#1
#2
#3
#4

probe

Cache Sets

S S S S S S S S S
R R R R R R R R R

receiver eviction sets
Jamming agreement

sender eviction sets

#1
#2
#3
#4

Cache Sets

S S S S S S S S S
R R R R R R R R R

receiver eviction sets

probe
Jamming agreement

sender eviction sets

#1
#2
#3
#4

prime

receive eviction sets

Cache Sets

S S S S S S S S S

S
Jamming agreement

sender eviction sets

#1
#2
#3
#4

receiver eviction sets

Cache Sets

S S S S S S S S
R R R R R R R R

prime
Jamming agreement

sender eviction sets

#1
#2
#3
#4

probe

Cache Sets

S S S S S S S S S

R R R R R R R R R

receiver eviction sets
Jamming agreement

sender eviction sets

#1
#2
#3
#4

Cache Sets

S S S S S S S S S
R R R R R R R R R

receiver eviction sets

probe
Jamming agreement

sender eviction sets

#1
#2
#3
#4

prime

receiver eviction sets

Cache Sets

S S S S S S S S
Jamming agreement

sender eviction sets

receiver eviction sets

Cache Sets

R R R R R R R R
S S S S S S S S S
Jamming agreement

sender eviction sets

receiver eviction sets

probe

#1
#2
#3
#4

Cache Sets

R R R R R R R R R
S S S S S S S S S

45
Jamming agreement

sender eviction sets

receiver eviction sets

Cache Sets

probe
Jamming agreement

sender eviction sets

#1
#2
#3
#4

prime

Cache Sets

S S S S S S S S S

receiver eviction sets
Jamming agreement

Cache Sets

sender eviction sets

receiver eviction sets

#1
#2
#3
#4

probe
Jamming agreement

sender eviction sets

#1
#2
#3
#4

Cache Sets

receiver eviction sets

#1
Jamming agreement

sender eviction sets

#1
#2
#3
#4

Cache Sets

receiver eviction sets

#1
### Jamming agreement

#### Sender eviction sets

<table>
<thead>
<tr>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Eviction Set #1" /></td>
<td><img src="image2" alt="Eviction Set #2" /></td>
<td><img src="image3" alt="Eviction Set #3" /></td>
<td><img src="image4" alt="Eviction Set #4" /></td>
</tr>
</tbody>
</table>

#### Receiver eviction sets

<table>
<thead>
<tr>
<th>#1</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Eviction Set #1" /></td>
</tr>
</tbody>
</table>

- **Repeat!**

---

45
Jamming agreement

sender eviction sets

#1
#2
#3
#4

receiver eviction sets

#1
#2

repeat!
Jamming agreement

sender eviction sets

#1
#2
#3
#4

receiver eviction sets

#1
#2
#3

repeat!
Jamming agreement

sender eviction sets

receiver eviction sets

#1  
#2  
#3  
#4

#1  
#2  
#3  
#4

repeat!
Sending the first image
Handling synchronization errors

- **deletion errors:** request-to-send scheme that also serves as ack
  - 3-bit sequence number
  - request: encoded sequence number (7 bits)

- **'0'-insertion errors:** error detection code
  - Berger codes
    - appending the number of '0's in the word to itself
    - property: a word cannot consist solely of '0's

**Physical layer word**

- 12 bits
- **Data**
Handling synchronization errors

- deletion errors: request-to-send scheme that also serves as ack
  - 3-bit sequence number
  - request: encoded sequence number (7 bits)

![Physical layer word diagram]

- Physical layer word:
  - Data: 12 bits
  - SQN: 3 bits
Handling synchronization errors

- deletion errors: request-to-send scheme that also serves as ack
  - 3-bit sequence number
  - request: encoded sequence number (7 bits)
- ’0’-insertion errors: error detection code → Berger codes
  - appending the number of ’0’s in the word to itself
  → property: a word cannot consist solely of ’0’s
Synchronization (before)
Synchronization (after)
Data-link layer: Error correction

- Reed-Solomon codes to correct the remaining errors
Data-link layer: Error correction

- Reed-Solomon codes to correct the remaining errors
- RS word size = physical layer word size = 12 bits
- packet size = $2^{12} - 1 = 4095$ RS words
- 10% error-correcting code: 409 parity and 3686 data RS words
Error correction (after)
### Evaluation

<table>
<thead>
<tr>
<th>Environment</th>
<th>Bit rate</th>
<th>Error rate</th>
<th>Noise</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native</td>
<td>75.10 KBps</td>
<td>0.00%</td>
<td>–</td>
</tr>
<tr>
<td>Environment</td>
<td>Bit rate</td>
<td>Error rate</td>
<td>Noise</td>
</tr>
<tr>
<td>-------------</td>
<td>------------</td>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Native</td>
<td>75.10 KBps</td>
<td>0.00%</td>
<td>–</td>
</tr>
<tr>
<td>Native</td>
<td>36.03 KBps</td>
<td>0.00%</td>
<td><strong>stress -m 1</strong></td>
</tr>
</tbody>
</table>
## Evaluation

<table>
<thead>
<tr>
<th>Environment</th>
<th>Bit rate</th>
<th>Error rate</th>
<th>Noise</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native</td>
<td>75.10 KBps</td>
<td>0.00%</td>
<td>–</td>
</tr>
<tr>
<td>Native</td>
<td>36.03 KBps</td>
<td>0.00%</td>
<td>stress -m 1</td>
</tr>
<tr>
<td>Amazon EC2</td>
<td>45.25 KBps</td>
<td>0.00%</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>stress -m 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>web server serving files on sender VM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>stress -m 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>stress -m 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>stress -m 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>stress -m 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>stress -m 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>web server on all 3 VMs, stress -m 4 on 3rd VM, stress -m 1 on sender and receiver VMs</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>stress -m 8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>stress -m 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>stress -m 8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>stress -m 1</td>
</tr>
</tbody>
</table>
### Evaluation

<table>
<thead>
<tr>
<th>Environment</th>
<th>Bit rate</th>
<th>Error rate</th>
<th>Noise</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native</td>
<td>75.10 KBps</td>
<td>0.00%</td>
<td>-</td>
</tr>
<tr>
<td>Native</td>
<td>36.03 KBps</td>
<td>0.00%</td>
<td>stress -m 1</td>
</tr>
<tr>
<td>Amazon EC2</td>
<td>45.25 KBps</td>
<td>0.00%</td>
<td>-</td>
</tr>
<tr>
<td>Amazon EC2</td>
<td>45.09 KBps</td>
<td>0.00%</td>
<td>web server serving files on sender VM</td>
</tr>
<tr>
<td>Amazon EC2</td>
<td>42.96 KBps</td>
<td>0.00%</td>
<td>stress -m 2 on sender VM</td>
</tr>
<tr>
<td>Amazon EC2</td>
<td>42.26 KBps</td>
<td>0.00%</td>
<td>stress -m 1 on receiver VM</td>
</tr>
<tr>
<td>Amazon EC2</td>
<td>37.42 KBps</td>
<td>0.00%</td>
<td>web server on all 3 VMs, stress -m 4 on 3rd VM, stress -m 1 on sender and receiver VMs</td>
</tr>
<tr>
<td>Amazon EC2</td>
<td>34.27 KBps</td>
<td>0.00%</td>
<td>stress -m 8 on third VM</td>
</tr>
</tbody>
</table>
Building an SSH connection

VM 1
- TCP Client (e.g. ssh)
- Socket
- TCP↔File
- File System
- Covert Channel

VM 2
- TCP Server (e.g. sshd)
- Socket
- TCP↔File
- File System
- Covert Channel

Hypervisor
- Prime+Probe

Last Level Cache (LLC)
### SSH evaluation

Between two instances on Amazon EC2

<table>
<thead>
<tr>
<th>Noise</th>
<th>Connection</th>
</tr>
</thead>
<tbody>
<tr>
<td>No noise</td>
<td>✓</td>
</tr>
<tr>
<td><code>stress -m 8</code> on third VM</td>
<td>✓</td>
</tr>
<tr>
<td>Web server on third VM</td>
<td>✓</td>
</tr>
<tr>
<td>Web server on SSH server VM</td>
<td>✓</td>
</tr>
<tr>
<td>Web server on all VMs</td>
<td>✓</td>
</tr>
<tr>
<td><code>stress -m 1</code> on server side</td>
<td>unstable</td>
</tr>
</tbody>
</table>

Telnet also works with occasional corrupted bytes with `stress -m 1`.
SSH evaluation

Between two instances on Amazon EC2

<table>
<thead>
<tr>
<th>Noise</th>
<th>Connection</th>
</tr>
</thead>
<tbody>
<tr>
<td>No noise</td>
<td>✓</td>
</tr>
<tr>
<td><strong>stress</strong> -m 8 on third VM</td>
<td>✓</td>
</tr>
<tr>
<td>Web server on third VM</td>
<td>✓</td>
</tr>
<tr>
<td>Web server on SSH server VM</td>
<td>✓</td>
</tr>
<tr>
<td>Web server on all VMs</td>
<td>✓</td>
</tr>
<tr>
<td><strong>stress</strong> -m 1 on server side</td>
<td>unstable</td>
</tr>
</tbody>
</table>

Telnet also works with occasional corrupted bytes with **stress** -m 1
Cache covert channels: Take-away

- cache covert channels are practical
Cache covert channels: Take-away

- cache covert channels are **practical**
- even in the **cloud**, even in presence of extraordinary **noise**
Cache covert channels: Take-away

- cache covert channels are **practical**
- even in the **cloud**, even in presence of extraordinary **noise**
- our robust covert channel supports an **SSH** connection
• cache covert channels are **practical**
• even in the **cloud**, even in presence of extraordinary **noise**
• our robust covert channel supports an SSH connection
• we extended Amazon’s product portfolio :)

Cache covert channels: Take-away
Cache covert channels: Take-away

- cache covert channels are practical
- even in the cloud, even in presence of extraordinary noise
- our robust covert channel supports an SSH connection
- we extended Amazon’s product portfolio :)
Cache covert channels: Take-away

- cache covert channels are practical
- even in the cloud, even in presence of extraordinary noise
- our robust covert channel supports an SSH connection
- we extended Amazon’s product portfolio :)
#2. Side-Channel Attack on Crypto
Side-channel attack on AES T-Tables implementation

- AES T-Tables: fast software implementation

Side-channel attack on AES T-Tables implementation

- AES T-Tables: fast software implementation
- uses precomputed look-up tables

---

Side-channel attack on AES T-Tables implementation

- **AES T-Tables**: fast software implementation
- uses precomputed look-up tables
- one-round known-plaintext attack by Osvik et al.
  - $p$ plaintext and $k$ secret key
  - intermediate state $x^{(r)} = (x_{0}^{(r)}, \ldots, x_{15}^{(r)})$ at each round $r$
  - first round, accessed table indices are
  \[ x_{i}^{(0)} = p_{i} \oplus k_{i} \quad \text{for all } i = 0, \ldots, 15 \]

---

Side-channel attack on AES T-Tables implementation

- **AES T-Tables**: fast software implementation
- uses precomputed look-up tables
- one-round known-plaintext attack by Osvik et al.
  - $p$ plaintext and $k$ secret key
  - intermediate state $x^{(r)} = (x^{(r)}_0, \ldots, x^{(r)}_{15})$ at each round $r$
  - first round, accessed table indices are

$$x^{(0)}_i = p_i \oplus k_i \quad \text{for all } i = 0, \ldots, 15$$

→ recovering accessed table indices $\Rightarrow$ recovering the key

---

Side-channel attack on AES T-Tables implementation

- monitoring which T-Table entry is accessed ($k_0 = 0x00$)
Side-channel attack on AES T-Tables implementation

• it’s an old attack...

Side-channel attack on AES T-Tables implementation

• it’s an old attack…
• everything should be fixed by now…

Side-channel attack on AES T-Tables implementation

• it’s an old attack...
• everything should be fixed by now...
• Bouncy Castle on Android → default implementation uses T-Tables

More attacks

• Flush+Reload
  • RSA: 96.7% of secret key bits in a single signature (Yarom et al.)
  • AES: full key recovery in 30000 dec. (a few seconds) (Guelmezoglu et al.)

• Prime+Probe
  • El Gamal (sliding window): full key recovery in 12 min. (Liu et al.)

---

B. Gülmezoglu et al. “A Faster and More Realistic Flush+Reload Attack on AES”. In: COSADE’15. 2015
you should never write crypto with secret-dependent memory accesses
Side-channel attack on crypto: Take-away

- You should never write crypto with secret-dependent memory accesses.
- Most recent implementations are now protected against these attacks.
- E.g., really hard to use AES T-Tables by mistake on OpenSSL.
• you should never write crypto with secret-dependent memory accesses
• most recent implementations are now protected against these attacks
• e.g., really hard to use AES T-Tables by mistake on OpenSSL
• big issue is adoption and legacy code still running
#3. Side-Channel Attack on Keystroke Timings
• locating *event-dependent* memory access → Cache Template Attacks

---

Cache Template Attacks

- locating **event-dependent** memory access $\rightarrow$ Cache Template Attacks
- learning phase

---

Cache Template Attacks

- locating event-dependent memory access → Cache Template Attacks
- learning phase
  1. shared library or executable is mapped

Cache Template Attacks

- locating **event-dependent** memory access → Cache Template Attacks
- learning phase
  1. shared library or executable is mapped
  2. trigger an event while Flush+Reload one address

---

Cache Template Attacks

- locating event-dependent memory access → Cache Template Attacks
- learning phase
  1. shared library or executable is mapped
  2. trigger an event while Flush+Reload one address
     → cache hit: address used by the library/executable

Cache Template Attacks

- locating **event-dependent** memory access → Cache Template Attacks
- learning phase
  1. shared library or executable is mapped
  2. trigger an event while Flush+Reload one address
     → cache hit: address used by the library/executable
  3. repeat step 2 for every address

---

Which addresses to monitor?

- cache template matrix
  = how many cache hits for each pair (event, address)?
Which addresses to monitor?

- cache template matrix
  = how many cache hits for each pair (event, address)?
Which addresses to monitor?

- cache template matrix
  = how many cache hits for each pair (event, address)?
Spying on keystrokes

- Flush+Reload and Evict+Reload: fine-grained attacks → spy on keystrokes
- high-resolution timers → precise inter-keystroke timing
- next step: infer typed words with Hidden Markov Models

Differentiating events

- distinguishing between **different types of events** by monitoring access time

---

Side-channel attack on keystrokes: Take-away

- obtaining **precise timings** is easy
Side-channel attack on keystrokes: Take-away

- obtaining *precise timings* is easy
- further computations are needed to derive typed words
Side-channel attack on keystrokes: Take-away

- obtaining *precise timings* is easy
- further computations are needed to derive typed words
- but the attack also allows *distinguishing key groups*
Side-channel attack on keystrokes: Take-away

- obtaining **precise timings** is easy
- further computations are needed to derive typed words
- but the attack also allows **distinguishing key groups**
  → reduces search space for, e.g., **password retrieval**
#4. Side-Channel Attack on KASLR
**prefetch instructions**

**prefetch** fetches the line of data from memory containing the specified byte

6 **prefetch** instructions:

- **prefectht0**: suggests CPU to load data into L1
- **prefectht1**: suggests CPU to load data into L2
- **prefectht2**: suggests CPU to load data into L3
- **prefetchnta**: suggests CPU to load data for non-temporal access
- **prefetchw**: suggests CPU to load data with intention to write
- **prefetchwt1**: suggests CPU to load vector data with intention to write
NOTE

Using the PREFETCH instruction is recommended only if data does not fit in cache. Use of software prefetch should be limited to memory addresses that are managed or owned within the application context. Prefetching to addresses that are not mapped to physical pages can experience non-deterministic performance penalty. For example specifying a NULL pointer (0L) as address for a prefetch can cause long delays.
NOTE

Using the PREFETCH instruction is recommended only if data does not fit in cache.
NOTE

Using the PREFETCH instruction is recommended only if data does not fit in cache. Use of software prefetch should be limited to memory addresses that are managed or owned within the application context.
NOTE

Using the PREFETCH instruction is recommended only if data does not fit in cache. Use of software prefetch should be limited to memory addresses that are managed or owned within the application context. Prefetching to addresses that are not mapped to physical pages can experience non-deterministic performance penalty.

NOTE

Using the PREFETCH instruction is recommended only if data does not fit in cache. Use of software prefetch should be limited to memory addresses that are managed or owned within the application context. Prefetching to addresses that are not mapped to physical pages can experience non-deterministic performance penalty. For example specifying a NULL pointer (0L) as address for a prefetch can cause long delays.

A little bit more background before continuing...
Address translation

- CR3
  - PML4
    - PML4E 0
    - PML4E 1
    - ...
    - #PML4I
    - PML4E 511
  - PDPT
    - PDPTI 0
    - PDPTI 1
    - ...
    - #PDPTI
    - PDPTI 511
    - Page Directory
      - PAGE 0
      - PAGE 1
      - ...
      - PAGE #PDI
      - PAGE 511
    - Page Table
      - PTE 0
      - PTE 1
      - ...
      - PTE #PTI
      - PTE 511
    - 4 KiB Page
      - Byte 0
      - Byte 1
      - ...
      - Offset
      - Byte 4095

48-bit virtual address

PML4I (9 b)  PDPTI (9 b)  PDI (9 b)  PTI (9 b)  Offset (12 b)
Address translation caches

Core 0
- ITLB
- DTLB
- PDE cache
- PDPTPE cache
- PML4E cache

Core 1
- ITLB
- DTLB
- PDE cache
- PDPTPE cache
- PML4E cache

Page table structures in system memory (DRAM)
Kernel is mapped in every process

Today’s operating systems:

Shared address space

User memory          Kernel memory

context switch
Kernel Address Space Layout Randomization (KASLR)

• same driver, different offset at each boot
Kernel Address Space Layout Randomization (KASLR)

- same driver, **different offset** at each boot
- leaking kernel/driver addresses defeats KASLR
Kernel direct-physical map

- OS X, Linux, BSD, Xen PVM (Amazon EC2)
Kernel direct-physical map

- OS X, Linux, BSD, Xen PVM (Amazon EC2)
- not Windows
Let’s go back to **prefetch**!
• tells the CPU “I might need that later”
prefetch: Unusual instructions (1)

- tells the CPU “I might need that later”
- hint—may be ignored by the CPU

• tells the CPU “I might need that later”
• hint—may be ignored by the CPU
• generates no faults

prefetch: Unusual instructions (1)

• tells the CPU “I might need that later”
• hint—may be ignored by the CPU
• generates no faults

Property #1: do not check privileges

• operand is a virtual address
prefetch: Unusual instructions (2)

- operand is a virtual address
- but it needs to translate the virtual address to a physical address

prefetch: Unusual instructions (2)

- operand is a virtual address
- but it needs to translate the virtual address to a physical address

Property #2: execution time varies

---

Exploiting property #1 + kernel direct-physical map

- Physical memory
- Virtual address space
- Cache

- Address-translation oracle
- Exploiting property #1 + kernel direct-physical map
- Virtual address space
- Physical memory
- Cache

- max. phys.
- direct map
- cache hit
Exploiting property #1 + kernel direct-physical map
Address-translation oracle

Exploiting property #1 + kernel direct-physical map
Address-translation oracle

Exploiting property #1 + kernel direct-physical map
Exploiting property #1 + kernel direct-physical map

- cache hit $\rightarrow$ physical address in kernel mapping is the correct translation
Translation-level oracle

Exploiting property #2

![Bar chart showing execution time in cycles for different mapping levels: PDPT, PD, PT, cached P., and uncached P. with values: 230, 246, 222, 181, and 383 respectively.]

• timing depends on where the translation stops
Translation-level oracle

Exploiting property #2

- timing depends on where the translation stops
Prefetch side-channel attacks

Using the two oracles

- variants of cache attacks (e.g., Flush+Prefetch)
- Rowhammer attacks on privileged addresses
- recovering translation levels of a process (/proc/pid/pagemap)
- now privileged bypasses ASLR
- translating virtual addresses to physical addresses (/proc/pid/pagemap)
- now privileged re-enables ret2dir exploits
- locating kernel drivers bypasses KASLR
Prefetch side-channel attacks

Using the two oracles

- variants of cache attacks (e.g., Flush+Prefetch)

Rowhammer attacks on privileged addresses

- recovering translation levels of a process
  - `/proc/pid/pagemap`

now privileged

re-enables

ret2dir exploits

- locating kernel drivers
  - bypasses KASLR
Prefetch side-channel attacks

Using the two oracles

• variants of cache attacks (e.g., Flush+Prefetch)
• Rowhammer attacks on privileged addresses
Prefetch side-channel attacks

Using the two oracles

- variants of cache attacks (e.g., Flush+Prefetch)
- Rowhammer attacks on privileged addresses
- recovering translation levels of a process (→ `/proc/pid/pagemap`)
  → now privileged → bypasses ASLR
Prefetch side-channel attacks

Using the two oracles

- variants of cache attacks (e.g., Flush+Prefetch)
- Rowhammer attacks on privileged addresses
- recovering translation levels of a process (→ /proc/pid/pagemap) → now privileged → bypasses ASLR
- translating virtual addresses to physical addresses (→ /proc/pid/pagemap) → now privileged → re-enables ret2dir exploits
Prefetch side-channel attacks

Using the two oracles

• variants of cache attacks (e.g., Flush+Prefetch)
• Rowhammer attacks on privileged addresses
• recovering translation levels of a process \(\rightarrow /\text{proc/pid/pagemap}\)  
  \(\rightarrow\) now privileged \(\rightarrow\) bypasses ASLR

• translating virtual addresses to physical addresses \(\rightarrow /\text{proc/pid/pagemap}\)  
  \(\rightarrow\) now privileged \(\rightarrow\) re-enables \text{ret2dir} exploits

• locating kernel drivers
  \(\rightarrow\) bypasses KASLR
For all mapped pages, found with the translation-level oracle
For all mapped pages, found with the translation-level oracle

1. evict translation caches: `Sleep()` / access large memory buffer
Defeating KASLR by locating kernel driver (1)

For all mapped pages, found with the translation-level oracle

1. evict translation caches: **Sleep()** / access large memory buffer
2. perform **syscall** to driver
Defeating KASLR by locating kernel driver (1)

For all mapped pages, found with the translation-level oracle

1. **evict** translation caches: `Sleep() / access large memory buffer`
2. perform **syscall** to driver
3. **time prefetch(page address)**
For all mapped pages, found with the translation-level oracle

1. **evict** translation caches: `Sleep()` / access large memory buffer
2. perform **syscall** to driver
3. **time** `prefetch(page address)`

→ fastest average access time is a driver page
Defeating KASLR by locating kernel driver (1)

For all mapped pages, found with the translation-level oracle

1. evict translation caches: `Sleep()` / access large memory buffer
2. perform syscall to driver
3. time `prefetch(page address)`
   → fastest average access time is a driver page

Full attack on Windows 10 in < 12 seconds
Defeating KASLR by locating kernel driver (2)
Prefetch side-channel attacks: Take-away

- **prefetch** does not allow you to **access memory**

---

Prefetch side-channel attacks: Take-away

- `prefetch` does not allow you to access memory
- doesn’t mean that it is a good idea to not perform any check

---

Prefetch side-channel attacks: Take-away

- **prefetch** does not allow you to access memory
- doesn’t mean that it is a good idea to not perform any check
- other microarchitectural attacks targeted KASLR

---

Prefetch side-channel attacks: Take-away

- **prefetch** does not allow you to access memory
- doesn’t mean that it is a good idea to not perform any check
- other microarchitectural attacks targeted KASLR
- still not completely dead

---

#5. Fault Attacks on DRAM in JavaScript
DRAM organization
DRAM organization

channel 0

channel 1
DRAM organization

channel 0

channel 1

back of DIMM: rank 1

front of DIMM: rank 0
DRAM organization

channel 0

channel 1

back of DIMM: rank 1

front of DIMM: rank 0

chip
### DRAM Organization

- **Chip**
- **Bank 0**
  - Row 0
  - Row 1
  - Row 2
  - ...
  - Row 32767
- **Row Buffer**

- **Bits in Cells in Rows**
- **Access:** activate row, copy to row buffer
DRAM refresh

- cells leak $\rightarrow$ repetitive **refresh** necessary
- refresh $\approx$ reading (destructive) + writing same data again
- maximum interval between refreshes to guarantee **data integrity**
DRAM refresh

- cells leak $\rightarrow$ repetitive refresh necessary
- refresh $\approx$ reading (destructive) + writing same data again
- maximum interval between refreshes to guarantee data integrity
- cells leak faster upon proximate accesses $\rightarrow$ Rowhammer
“It’s like breaking into an apartment by repeatedly slamming a neighbor’s door until the vibrations open the door you were after” – Motherboard Vice
“It’s like breaking into an apartment by repeatedly slamming a neighbor’s door until the vibrations open the door you were after” – Motherboard Vice
“It’s like breaking into an apartment by repeatedly slamming a neighbor’s door until the vibrations open the door you were after” – Motherboard Vice
“It’s like breaking into an apartment by repeatedly slamming a neighbor’s door until the vibrations open the door you were after” – Motherboard Vice
“It’s like breaking into an apartment by repeatedly slamming a neighbor’s door until the vibrations open the door you were after” – Motherboard Vice
“It’s like breaking into an apartment by repeatedly slamming a neighbor’s door until the vibrations open the door you were after” – Motherboard Vice
Impact of the CPU cache

- only non-cached accesses reach DRAM
- original attacks use `clflush` instruction
  → flush line from cache
  → next access will be served from DRAM
Rowhammer (with clflush)
Rowhammer (with clflush)

- Cache set 1
- Cache set 2
- clflush

DRAM bank
Rowhammer (with clflush)
Rowhammer (with clflush)

- DRAM bank
- cache set 1
- cache set 2
Rowhammer (with clflush)

- Cache set 1
- Cache set 2
- DRAM bank

Reload
Rowhammer (with clflush)
Rowhammer (with clflush)

- Cache set 1
- Cache set 2
- DRAM bank

The diagram illustrates the interaction between cache sets and a DRAM bank, showing the impact of clflush operations on memory coherence.
Rowhammer (with clflush)

- Cache set 1
- Cache set 2
- DRAM bank

Reload arrows indicate the process of refreshing cache sets.
Rowhammer (with clflush)

cache set 1

cache set 2

clflush

clflush

DRAM bank
Rowhammer (with clflush)
Rowhammer (with clflush)
Rowhammer (with clflush)

- Cache set 1
- Cache set 2
- DRAM bank

reload

89
Rowhammer (with clflush)

Cache set 1

Cache set 2

DRAM bank

wait for it...

clflush

clflush

clflush
Rowhammer (with clflush)

Cache set 1

Cache set 2

DRAM bank

bit flip!

reload

reload
Flush, reload, flush, reload...

- the core of Rowhammer is essentially a Flush+Reload loop
- as much an attack on DRAM as on cache
Rowhammer without clflush?

- idea: avoid clflush to be independent of specific instructions
  - no clflush in JavaScript

---

Rowhammer without clflush?

- **idea**: avoid `clflush` to be independent of specific instructions
  - no `clflush` in JavaScript
- **our approach**: use regular memory accesses for eviction
  - techniques from cache attacks!

---

Rowhammer without clflush?

• idea: avoid clflush to be independent of specific instructions
  → no clflush in JavaScript

• our approach: use regular memory accesses for eviction
  → techniques from cache attacks!
  → Rowhammer, Prime+Probe style!

Rowhammer without clflush

cache set 1

cache set 2

DRAM bank
Rowhammer without clflush

cache set 1

load

load

cache set 2

DRAM bank
Rowhammer without clflush

cache set 1

cache set 2

DRAM bank
Rowhammer without clflush

cache set 1

cache set 2

DRAM bank

load

load
Rowhammer without clflush

cache set 1

cache set 2

DRAM bank
Rowhammer without `clflush`

cache set 1

cache set 2
Rowhammer without clflush

Cache set 1

Cache set 2

Load

DRAM bank
Rowhammer without clflush

Cache set 1

Cache set 2

DRAM bank
Rowhammer without clflush

cache set 1

load

DRAM bank

load

cache set 2
Rowhammer without clflush
Rowhammer without clflush

cache set 1

cache set 2

repeat!

DRAM bank
Rowhammer without clflush

cache set 1

cache set 2

DRAM bank

wait for it…
Rowhammer without clflush

Cache set 1

Cache set 2

DRAM bank

bit flip!
Replacement policy since Ivy Bridge

$n$ accesses for an $n$-way cache

- no LRU replacement

![Cache Set Diagram](image)
Replacement policy since Ivy Bridge

$n$ accesses for an $n$-way cache

- no LRU replacement
Replacement policy since Ivy Bridge

\( n \) accesses for an \( n \)-way cache

- no LRU replacement
Replacement policy since Ivy Bridge

$n$ accesses for an $n$-way cache

- no LRU replacement
Replacement policy since Ivy Bridge

$n$ accesses for an $n$-way cache

- no LRU replacement
Replacement policy since Ivy Bridge

\( n \) accesses for an \( n \)-way cache

- no LRU replacement
Replacement policy since Ivy Bridge

$n$ accesses for an $n$-way cache

- no LRU replacement
Replacement policy since Ivy Bridge

$n$ accesses for an $n$-way cache

- no LRU replacement
Replacement policy since Ivy Bridge

$n$ accesses for an $n$-way cache

- no LRU replacement

![Cache Set Diagram](image-url)
Replacement policy since Ivy Bridge

\(n\) accesses for an \(n\)-way cache

- no LRU replacement
- only 75% success rate on Haswell
Replacement policy since Ivy Bridge

\( n \) accesses for an \( n \)-way cache

- no LRU replacement
- only 75% success rate on Haswell
- more accesses → higher success rate, but too slow
Requirements for Rowhammer

1. **uncached** memory accesses: need to reach DRAM
2. **fast** memory accesses: race against the next row refresh
Requirements for Rowhammer

1. uncached memory accesses: need to reach DRAM
2. fast memory accesses: race against the next row refresh

→ optimize the eviction rate and the timing
Cache eviction strategies: The beginning

→ fast and effective on Haswell: eviction rate >99.97%
Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...

<table>
<thead>
<tr>
<th>strategy</th>
<th># accesses</th>
<th>eviction rate</th>
<th>loop time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{P}$-1-1-1-17</td>
<td>17</td>
<td></td>
<td>17</td>
</tr>
<tr>
<td>$\mathcal{P}$-1-1-1-20</td>
<td>20</td>
<td></td>
<td>20</td>
</tr>
</tbody>
</table>

Executed in a loop, on a Haswell with a 16-way last-level cache
Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...

<table>
<thead>
<tr>
<th>strategy</th>
<th># accesses</th>
<th>eviction rate</th>
<th>loop time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$-1-1-1-17</td>
<td>17</td>
<td>74.46%</td>
<td>❌</td>
</tr>
<tr>
<td>$P$-1-1-1-20</td>
<td>20</td>
<td>99.82%</td>
<td>✔️</td>
</tr>
</tbody>
</table>

Executed in a loop, on a Haswell with a 16-way last-level cache
Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...

<table>
<thead>
<tr>
<th>strategy</th>
<th># accesses</th>
<th>eviction rate</th>
<th>loop time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P-1-1-1-17</td>
<td>17</td>
<td>74.46% ⚹</td>
<td>307 ns ✓</td>
</tr>
<tr>
<td>P-1-1-1-20</td>
<td>20</td>
<td>99.82% ✓</td>
<td>934 ns ✗</td>
</tr>
</tbody>
</table>

Executed in a loop, on a Haswell with a 16-way last-level cache
Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...

<table>
<thead>
<tr>
<th>strategy</th>
<th># accesses</th>
<th>eviction rate</th>
<th>loop time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_{1-1-1-17}$</td>
<td>17</td>
<td>74.46%</td>
<td>✓</td>
</tr>
<tr>
<td>$P_{1-1-1-20}$</td>
<td>20</td>
<td>99.82%</td>
<td>✓</td>
</tr>
<tr>
<td>$P_{2-1-1-17}$</td>
<td>34</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Executed in a loop, on a Haswell with a 16-way last-level cache
We evaluated more than 10000 strategies...

<table>
<thead>
<tr>
<th>strategy</th>
<th># accesses</th>
<th>eviction rate</th>
<th>loop time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P-1-1-1-17</td>
<td>17</td>
<td>74.46% ❌</td>
<td>307 ns ✓</td>
</tr>
<tr>
<td>P-1-1-1-20</td>
<td>20</td>
<td>99.82% ✓</td>
<td>934 ns ❌</td>
</tr>
<tr>
<td>P-2-1-1-17</td>
<td>34</td>
<td>99.86% ✓</td>
<td></td>
</tr>
</tbody>
</table>

Executed in a loop, on a Haswell with a 16-way last-level cache
We evaluated more than 10000 strategies...

<table>
<thead>
<tr>
<th>strategy</th>
<th># accesses</th>
<th>eviction rate</th>
<th>loop time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P-1-1-1-17</td>
<td>17</td>
<td>74.46%</td>
<td>307 ns</td>
</tr>
<tr>
<td>P-1-1-1-20</td>
<td>20</td>
<td>99.82%</td>
<td>934 ns</td>
</tr>
<tr>
<td>P-2-1-1-17</td>
<td>34</td>
<td>99.86%</td>
<td>191 ns</td>
</tr>
</tbody>
</table>

Executed in a loop, on a Haswell with a 16-way last-level cache
We evaluated more than 10000 strategies...

<table>
<thead>
<tr>
<th>strategy</th>
<th># accesses</th>
<th>eviction rate</th>
<th>loop time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P-1-1-1-17</td>
<td>17</td>
<td>74.46% ⚹️</td>
<td>307 ns ✔️</td>
</tr>
<tr>
<td>P-1-1-1-20</td>
<td>20</td>
<td>99.82% ✔️</td>
<td>934 ns ⚹️</td>
</tr>
<tr>
<td>P-2-1-1-17</td>
<td>34</td>
<td>99.86% ✔️</td>
<td>191 ns ✔️</td>
</tr>
<tr>
<td>P-2-2-1-17</td>
<td>64</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Executed in a loop, on a Haswell with a 16-way last-level cache
Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...

<table>
<thead>
<tr>
<th>strategy</th>
<th># accesses</th>
<th>eviction rate</th>
<th>loop time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$-1-1-1-17</td>
<td>17</td>
<td>74.46% ✗</td>
<td>307 ns ✔</td>
</tr>
<tr>
<td>$P$-1-1-1-20</td>
<td>20</td>
<td>99.82% ✔</td>
<td>934 ns ✗</td>
</tr>
<tr>
<td>$P$-2-1-1-17</td>
<td>34</td>
<td>99.86% ✔</td>
<td>191 ns ✔</td>
</tr>
<tr>
<td>$P$-2-2-1-17</td>
<td>64</td>
<td>99.98% ✔</td>
<td></td>
</tr>
</tbody>
</table>

Executed in a loop, on a Haswell with a 16-way last-level cache
Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...

<table>
<thead>
<tr>
<th>strategy</th>
<th># accesses</th>
<th>eviction rate</th>
<th>loop time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$-1-1-1-17</td>
<td>17</td>
<td>74.46%</td>
<td>307 ns ✓</td>
</tr>
<tr>
<td>$P$-1-1-1-20</td>
<td>20</td>
<td>99.82% ✓</td>
<td>934 ns ×</td>
</tr>
<tr>
<td>$P$-2-1-1-17</td>
<td>34</td>
<td>99.86% ✓</td>
<td>191 ns ✓</td>
</tr>
<tr>
<td>$P$-2-2-1-17</td>
<td>64</td>
<td>99.98% ✓</td>
<td>180 ns ✓</td>
</tr>
</tbody>
</table>

Executed in a loop, on a Haswell with a 16-way last-level cache
### Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...

<table>
<thead>
<tr>
<th>strategy</th>
<th># accesses</th>
<th>eviction rate</th>
<th>loop time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1-1-17</td>
<td>17</td>
<td>74.46%</td>
<td>307 ns</td>
</tr>
<tr>
<td>1-1-1-20</td>
<td>20</td>
<td>99.82%</td>
<td>934 ns</td>
</tr>
<tr>
<td>2-1-1-17</td>
<td>34</td>
<td>99.86%</td>
<td>191 ns</td>
</tr>
<tr>
<td>2-2-1-17</td>
<td>64</td>
<td>99.98%</td>
<td>180 ns</td>
</tr>
</tbody>
</table>

→ more accesses, smaller execution time?

Executed in a loop, on a Haswell with a 16-way last-level cache.
Cache eviction strategies: Illustration

\( \mathcal{P}-1-1-17 \) (17 accesses, 307ns)

\( \mathcal{P}-2-1-17 \) (34 accesses, 191ns)
Cache eviction strategies: Illustration

$\mathcal{P}-1-1-1-17$ (17 accesses, 307ns)

$\mathcal{P}-2-1-1-17$ (34 accesses, 191ns)
Cache eviction strategies: Illustration

\( \mathcal{P} \)-1-1-1-17 (17 accesses, 307ns)

\( \mathcal{P} \)-2-1-1-17 (34 accesses, 191ns)
Cache eviction strategies: Illustration

$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)

$\mathcal{P}$-2-1-1-17 (34 accesses, 191ns)
Cache eviction strategies: Illustration

\( \mathcal{P}-1-1-1-17 \) (17 accesses, 307ns)

\( \mathcal{P}-2-1-1-17 \) (34 accesses, 191ns)
Cache eviction strategies: Illustration

\( \mathcal{P}-1-1-1-17 \) (17 accesses, 307ns)

\( \mathcal{P}-2-1-1-17 \) (34 accesses, 191ns)
Cache eviction strategies: Illustration

$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)

$\mathcal{P}$-2-1-1-17 (34 accesses, 191ns)

Time in ns
Cache eviction strategies: Illustration

\( \mathcal{P} \)-1-1-1-17 (17 accesses, 307ns)

\( \mathcal{P} \)-2-1-1-17 (34 accesses, 191ns)
Cache eviction strategies: Illustration

\[\mathcal{P}-1-1-1-17\] (17 accesses, 307ns)

\[\mathcal{P}-2-1-1-17\] (34 accesses, 191ns)
Cache eviction strategies: Illustration

\(P-1-1-1-17\) (17 accesses, 307ns)

\(P-2-1-1-17\) (34 accesses, 191ns)

Time in ns
Cache eviction strategies: Illustration

\( P-1-1-1-17 \) (17 accesses, 307ns)

\( P-2-1-1-17 \) (34 accesses, 191ns)

Time in ns
Cache eviction strategies: Illustration

$P$-1-1-1-17 (17 accesses, 307ns)

$P$-2-1-1-17 (34 accesses, 191ns)
Cache eviction strategies: Illustration

$P$-1-1-1-17 (17 accesses, 307ns)

$P$-2-1-1-17 (34 accesses, 191ns)

Time in ns
Cache eviction strategies: Illustration

\( \mathcal{P}-1-1-1-17 \) (17 accesses, 307ns)

\( \mathcal{P}-2-1-1-17 \) (34 accesses, 191ns)
Cache eviction strategies: Illustration

\( P-1-1-1-17 \) (17 accesses, 307ns)

\( P-2-1-1-17 \) (34 accesses, 191ns)
Cache eviction strategies: Illustration

$P_{1-1-1-17}$ (17 accesses, 307ns)

$P_{2-1-1-17}$ (34 accesses, 191ns)
Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)
Cache eviction strategies: Illustration

\[P-1-1-1-17\] (17 accesses, 307ns)

\[P-2-1-1-17\] (34 accesses, 191ns)

Time in ns
Cache eviction strategies: Illustration

$\mathcal{P}-1-1-1-17$ (17 accesses, 307ns)

$\mathcal{P}-2-1-1-17$ (34 accesses, 191ns)
Cache eviction strategies: Illustration

\( P-1-1-1-17 \) (17 accesses, 307ns)

\( P-2-1-1-17 \) (34 accesses, 191ns)

Time in ns
Cache eviction strategies: Illustration

\(P-1-1-1-17\) (17 accesses, 307ns)

\(P-2-1-1-17\) (34 accesses, 191ns)

Time in ns
\( \mathcal{P}-1-1-1-17 \) (17 accesses, 307ns)

\( \mathcal{P}-2-1-1-17 \) (34 accesses, 191ns)
Cache eviction strategies: Illustration

\( P-1-1-1-17 \) (17 accesses, 307ns)

\( P-2-1-1-17 \) (34 accesses, 191ns)

Time in ns
Cache eviction strategies: Illustration

\( P-1-1-1-17 \) (17 accesses, 307ns)

\( P-2-1-1-17 \) (34 accesses, 191ns)

Time in ns
Cache eviction strategies: Illustration

\( \mathcal{P}-1-1-1-17 \) (17 accesses, 307ns)

\( \mathcal{P}-2-1-1-17 \) (34 accesses, 191ns)
Cache eviction strategies: Illustration

\[ -1-1-1-17 \text{ (17 accesses, 307ns)} \]

\[ -2-1-1-17 \text{ (34 accesses, 191ns)} \]
Cache eviction strategies: Illustration

\(P-1-1-1-17\) (17 accesses, 307ns)

\(P-2-1-1-17\) (34 accesses, 191ns)
Cache eviction strategies: Illustration

\(\mathcal{P}-1-1-1-17\) (17 accesses, 307ns)

\(\mathcal{P}-2-1-1-17\) (34 accesses, 191ns)
Cache eviction strategies: Illustration

\( P-1-1-1-17 \) (17 accesses, 307ns)

\( P-2-1-1-17 \) (34 accesses, 191ns)

Time in ns
Cache eviction strategies: Illustration

\[ \mathcal{P}-1-1-1-17 \text{ (17 accesses, 307ns)} \]

\[ \mathcal{P}-2-1-1-17 \text{ (34 accesses, 191ns)} \]
Cache eviction strategies: Illustration

\( \mathcal{P}-1-1-1-17 \) (17 accesses, 307ns)

\( \mathcal{P}-2-1-1-17 \) (34 accesses, 191ns)

Time in ns
Cache eviction strategies: Illustration

\[ P-1-1-1-17 \] (17 accesses, 307ns)

\[ P-2-1-1-17 \] (34 accesses, 191ns)
Cache eviction strategies: Illustration

\( \mathcal{P}-1-1-1-17 \) (17 accesses, 307ns)

<table>
<thead>
<tr>
<th>Miss (intended)</th>
<th>Miss (intended)</th>
<th>Miss</th>
<th>Miss</th>
<th>Miss</th>
<th>Miss</th>
<th>Miss</th>
<th>Miss</th>
</tr>
</thead>
</table>

\( \mathcal{P}-2-1-1-17 \) (34 accesses, 191ns)

<table>
<thead>
<tr>
<th>Miss (intended)</th>
<th>Miss (intended)</th>
<th>Miss</th>
<th>Miss</th>
<th>Miss</th>
<th>Miss</th>
<th>Miss</th>
<th>Miss</th>
</tr>
</thead>
</table>

Time in ns
Cache eviction strategies: Illustration

\[ \mathcal{P}-1-1-1-17 \] (17 accesses, 307ns)

\[ \mathcal{P}-2-1-1-17 \] (34 accesses, 191ns)

Time in ns
Cache eviction strategies: Illustration

\( \mathcal{P}-1-1-1-17 \) (17 accesses, 307ns)

\( \mathcal{P}-2-1-1-17 \) (34 accesses, 191ns)

Time in ns
Cache eviction strategies: Illustration

\(P-1-1-1-17\) (17 accesses, 307ns)

\(P-2-1-1-17\) (34 accesses, 191ns)
Cache eviction strategies: Illustration

\[ P-1-1-1-17 \text{ (17 accesses, 307ns)} \]

\[ P-2-1-1-17 \text{ (34 accesses, 191ns)} \]
Cache eviction strategies: Illustration

\( \mathcal{P}-1-1-1-17 \) (17 accesses, 307ns)

\( \mathcal{P}-2-1-1-17 \) (34 accesses, 191ns)

Time in ns
Cache eviction strategies: Illustration

\[ \mathcal{P}-1-1-1-17 \] (17 accesses, 307ns)

\[ \mathcal{P}-2-1-1-17 \] (34 accesses, 191ns)
Cache eviction strategies: Illustration

\( \mathcal{P}-1-1-1-17 \) (17 accesses, 307ns)

\( \mathcal{P}-2-1-1-17 \) (34 accesses, 191ns)
Cache eviction strategies: Illustration

\( P-1-1-1-17 \) (17 accesses, 307ns)

\( P-2-1-1-17 \) (34 accesses, 191ns)
Cache eviction strategies: Illustration

\( \mathcal{P}-1-1-1-17 \) (17 accesses, 307ns)

\( \mathcal{P}-2-1-1-17 \) (34 accesses, 191ns)
Cache eviction strategies: Illustration

$P$-1-1-1-17 (17 accesses, 307ns)

$P$-2-1-1-17 (34 accesses, 191ns)
Evaluation on Haswell

Number of bit flips within 15 minutes.

- clflush
- Evict (Native)
- Evict (JavaScript)
Rowhammer.js: Take-Away

- cache eviction fast enough to replace \texttt{clflush}
- independent of programming language and available instructions

---

Rowhammer.js: Take-Away

- cache eviction fast enough to replace \texttt{clflush}
- independent of programming language and available instructions
- lessons learned from side channels to perform a fault attack

---

Rowhammer.js: Take-Away

- cache eviction fast enough to replace `clflush`
- independent of programming language and available instructions
- lessons learned from side channels to perform a fault attack
- first remote fault attack, from a browser

---

Rowhammer.js: Take-Away

- cache eviction fast enough to replace `clflush`
- independent of programming language and available instructions
- lessons learned from side channels to perform a fault attack
- first remote fault attack, from a browser
- if you think a fault is not exploitable, think again

Countermeasures
Countermeasures

- different levels: hardware, system, application
- different goals
  - remove interferences
  - add noise to interferences
  - make it impossible to measure interferences
Hardware level: Fixing the instruction set?

- clflush
  - unprivileged line eviction
Hardware level: Fixing the instruction set?

- **clflush**
  - unprivileged line eviction → make it **privileged**

require changes to the architecture; attacks still possible (e.g., Prime+Probe)
Hardware level: Fixing the instruction set?

- **clflush**
  - unprivileged line eviction → make it privileged
  - leaks timing information
Hardware level: Fixing the instruction set?

- **clflush**
  - unprivileged line eviction → make it **privileged**
  - leaks timing information → make it **constant-time**
Hardware level: Fixing the instruction set?

- **clflush**
  - unprivileged line eviction → make it **privileged**
  - leaks timing information → make it **constant-time**

- **rdtsc**
  - unprivileged fine-grained timing
Hardware level: Fixing the instruction set?

- **clflush**
  - unprivileged line eviction → make it privileged
  - leaks timing information → make it constant-time

- **rdtsc**
  - unprivileged fine-grained timing → make it privileged
Hardware level: Fixing the instruction set?

- **clflush**
  - unprivileged line eviction → make it privileged
  - leaks timing information → make it constant-time

- **rdtsc**
  - unprivileged fine-grained timing → make it privileged

→ require changes to the architecture
Hardware level: Fixing the instruction set?

- **clflush**
  - unprivileged line eviction → make it privileged
  - leaks timing information → make it constant-time

- **rdtsc**
  - unprivileged fine-grained timing → make it privileged

→ require changes to the architecture
→ attacks still possible (e.g., Prime+Probe)
Hardware level: Stop sharing hardware?

- stop sharing cache

---


Hardware level: Stop sharing hardware?

- **stop sharing cache** → attacks are getting better
  - first attacks on L1 → same core

---


Hardware level: Stop sharing hardware?

- stop sharing cache → attacks are getting better
  - first attacks on L1 → same core → stop sharing core

---


Hardware level: Stop sharing hardware?

- **stop sharing cache** → attacks are getting better
  - first attacks on L1 → same core → stop sharing core
  - current attacks on LLC → across cores

---


Hardware level: Stop sharing hardware?

- **stop sharing cache** → attacks are getting better
  - first attacks on L1 → same core → stop sharing core
  - current attacks on LLC → across cores → stop sharing CPU

---


Hardware level: Stop sharing hardware?

- stop sharing cache → attacks are getting better
  - first attacks on L1 → same core → stop sharing core
  - current attacks on LLC → across cores → stop sharing CPU
  - 2016: first attack across processors

---


Hardware level: Stop sharing hardware?

- stop sharing cache → attacks are getting better
  - first attacks on L1 → same core → stop sharing core
  - current attacks on LLC → across cores → stop sharing CPU
  - 2016: first attack across processors → what next?

---

Hardware level: Stop sharing hardware?

- **stop sharing cache** → attacks are getting better
  - first attacks on L1 → same core → stop sharing core
  - current attacks on LLC → across cores → stop sharing CPU
  - 2016: first attack across processors → what next?

- **not an option** for cost reasons in the cloud

---


Hardware level: Stop sharing hardware?

- **stop sharing cache** → attacks are getting better
  - first attacks on L1 → same core → stop sharing core
  - current attacks on LLC → across cores → stop sharing CPU
  - 2016: first attack across processors → what next?

- **not an option** for cost reasons in the cloud

- what about JavaScript attacks?

---


Hardware level: Changes in microarchitecture

- secure cache designs: Random-Permutation Cache, Partition-Locked Cache

\[\text{References}\]


J. Kong et al. “Hardware-software integrated approaches to defend against software cache-based side channel attacks”. In: HPCA’09. 2009.

Hardware level: Changes in microarchitecture

- secure cache designs: Random-Permutation Cache, Partition-Locked Cache
  → expensive, not always high performance

---


J. Kong et al. “Hardware-software integrated approaches to defend against software cache-based side channel attacks”. In: HPCA’09. 2009.

Hardware level: Changes in microarchitecture

- secure cache designs: Random-Permutation Cache, Partition-Locked Cache → expensive, not always high performance
- disruptive prefetching: random hardware prefetches

---


J. Kong et al. “Hardware-software integrated approaches to defend against software cache-based side channel attacks”. In: HPCA’09. 2009.

Hardware level: Changes in microarchitecture

- secure cache designs: Random-Permutation Cache, Partition-Locked Cache
  → expensive, not always high performance
- disruptive prefetching: random hardware prefetches
  → adding noise makes attacks harder, not impossible

---


Hardware level: Changes in microarchitecture

- secure cache designs: Random-Permutation Cache, Partition-Locked Cache
  - expensive, not always high performance
- disruptive prefetching: random hardware prefetches
  - adding noise makes attacks harder, not impossible
  - trade-off security/performance/cost

---


J. Kong et al. “Hardware-software integrated approaches to defend against software cache-based side channel attacks”. In: HPCA'09. 2009.

Hardware level: Changes in microarchitecture

- secure cache designs: Random-Permutation Cache, Partition-Locked Cache
  → expensive, not always high performance
- disruptive prefetching: random hardware prefetches
  → adding noise makes attacks harder, not impossible
→ trade-off security/performance/cost
→ performance and cost win, no implementation by manufacturers

---


J. Kong et al. “Hardware-software integrated approaches to defend against software cache-based side channel attacks”. In: HPCA’09. 2009.

System level: Prevention

- L1 cache cleansing

Y. Zhang et al. “Dupe: retrofitting commodity operating systems to mitigate cache side channels in the cloud”. In: CCS’13. 2013
B. C. Vattikonda et al. “Eliminating fine grained timers in Xen”. In: CCSW’11. 2011
System level: Prevention

- L1 cache cleansing
  → if applied to LLC → same as no cache, disastrous performance

---

Y. Zhang et al. “Düppel: retrofitting commodity operating systems to mitigate cache side channels in the cloud”. In: CCS’13. 2013


B. C. Vattikonda et al. “Eliminating fine grained timers in Xen”. In: CCSW’11. 2011


104
System level: Prevention

- L1 cache cleansing
  - if applied to LLC → same as no cache, disastrous performance
- page coloring → reduces cache sharing

---

Y. Zhang et al. “Düppel: retrofitting commodity operating systems to mitigate cache side channels in the cloud”. In: CCS’13. 2013


B. C. Vattikonda et al. “Eliminating fine grained timers in Xen”. In: CCSW’11. 2011

System level: Prevention

• L1 cache cleansing
  → if applied to LLC → same as no cache, disastrous performance

• page coloring → reduces cache sharing
  → limited number of colors + bad performance
  → doesn’t prevent Flush+Reload

Y. Zhang et al. “Düppel: retrofitting commodity operating systems to mitigate cache side channels in the cloud”. In: CCS’13. 2013
B. C. Vattikonda et al. “Eliminating fine grained timers in Xen”. In: CCSW’11. 2011
System level: Prevention

- L1 cache cleansing
  - if applied to LLC → same as no cache, disastrous performance
- page coloring → reduces cache sharing
  - limited number of colors + bad performance
  - doesn’t prevent Flush+Reload
- noise in timers or no timer

Y. Zhang et al. “Düppel: retrofitting commodity operating systems to mitigate cache side channels in the cloud”. In: CCS’13. 2013
B. C. Vattikonda et al. “Eliminating fine grained timers in Xen”. In: CCSW’11. 2011
System level: Prevention

- L1 cache cleansing
  - if applied to LLC → same as no cache, disastrous performance

- page coloring → reduces cache sharing
  - limited number of colors + bad performance
  - doesn’t prevent Flush+Reload

- noise in timers or no timer
  - adding noise makes attacks harder, not impossible
  - removing timers is not realistic

Y. Zhang et al. “Düppel: retrofitting commodity operating systems to mitigate cache side channels in the cloud”. In: CCS’13. 2013
B. C. Vattikonda et al. “Eliminating fine grained timers in Xen”. In: CCSW’11. 2011
System level: Detect on-going attacks

- using **performance counters** to monitor cache hits and cache misses
  → risk of false positives


Application level: Detect leakage

- CacheAudit: static analysis of source code
- Cache Template Attacks: dynamic approach

---

Application level: Detect leakage

- CacheAudit: static analysis of source code
- Cache Template Attacks: dynamic approach

→ limited to side-channels → covert channels still possible
→ most effective for critical code

---


Application level: Detect leakage

- CacheAudit: **static analysis** of source code
- Cache Template Attacks: **dynamic approach**
  → limited to side-channels → covert channels still possible
  → most effective for critical code
  → see Boris’ talk this afternoon!

---

Application level: Write better code

- square-and-multiply-always algorithm
- bit-sliced AES implementation
- hardware implementations (AES-NI, etc.)
Application level: Write better code

- square-and-multiply-always algorithm
- bit-sliced AES implementation
- hardware implementations (AES-NI, etc.)

→ protecting crypto is possible and necessary!
→ a few CVEs that have been treated: CVE-2005-0109, CVE-2013-4242, CVE-2014-0076, CVE-2016-0702, CVE-2016-2178
Bigger Perspective and Conclusions
rdseed and floating point operations

- rdseed
  - request a random seed to the hardware random number generator
  - fixed number of precomputed random bits, takes time to regenerate them
  → covert channel

M. Andrysco et al. “On subnormal floating point and abnormal timing”. In: S&P’15. 2015
rdseed and floating point operations

- **rdseed**
  - request a random seed to the hardware random number generator
  - fixed number of precomputed random bits, takes time to regenerate them
  → covert channel

- **fadd, fmul**
  - floating-point unit
  - floating point operations running time depends on the operands
  → bypassing Firefox’s same origin policy via SVG filter timing attack

---


M. Andrysco et al. “On subnormal floating point and abnormal timing”. In: S&P’15. 2015
jmp and TSX instructions

- jmp
  - branch prediction and branch target prediction → branch prediction unit
  → covert channels, side-channel attacks on crypto, bypassing kernel ASLR

jmp and TSX instructions

- jmp
  - branch prediction and branch target prediction → branch prediction unit
    → covert channels, side-channel attacks on crypto, bypassing kernel ASLR

- TSX instructions
  - extension for transactional memory support in hardware
    → bypassing kernel ASLR

---

O. Acıçmez et al. "Predicting secret keys via branch prediction”. In: CT-RSA 2007. 2007
• more a problem of CPU design than Instruction Set Architecture
• more a problem of **CPU design** than Instruction Set Architecture
• hard to patch → issues linked to performance **optimizations**
Conclusion

• more a problem of **CPU design** than Instruction Set Architecture
• hard to patch → issues linked to performance **optimizations**
• quick fixes like removing instructions won’t work
• more a problem of *CPU design* than Instruction Set Architecture
• hard to patch → issues linked to performance *optimizations*
• quick fixes like removing instructions won’t work
→ looking forward to Boris’ talk this afternoon!
Questions?

Contact

✉️ clementine.maurice@iaik.tugraz.at
📍 @BloodyTangerine
Cache attacks:
Software side-channel and fault attacks

Clémentine Maurice, Graz University of Technology
May 11, 2017—Spring school on Security & Correctness in the IoT 2017