Microarchitectural Data Leakage

via Automated Attack Synthesis

Daniel Moghimi, Worcester Polytechnic University

Jun 23, 2020
Virtual Talk for Intel Product Security Incident Team (IPSIRT)
About Me

• Daniel Moghimi (@danielmgmi)
• Security Researcher
• PhD Student @ WPI
  • Microarchitectural Security
  • Side Channels
  • Breaking Crypto Implementations
  • Trusted Execution Environment (Intel SGX)

• Contributed to:
  • ZombieLoad, Fallout, LVI,
  • MemJam, Spoiler, CacheZoom, CopyCat
  • Jackhammer, TPM-Fail
Thanks...

• Berk Sunar @ WPI

• Moritz Lipp @ tugraz

• Michael Schwartz @ tugraz
Disclaimers

• Our findings and reasonings are based on:
  • RE
  • Patents
  • Analysis

• You may know more than me how Intel CPU works!!!
Today’s Agenda

• Motivation: Meltdown-style Attacks

• Background: CPU Memory Subsystem

• Transynther, Automated Attack Synthesis

• MDS Root Cause Analysis and new subvariants

• Medusa attack and RSA key recovery
2018: Meltdown Attack?

```c
char secret = *(char *) 0xffffffff81a0123;
printf("%c\n", secret);
```
2018: Meltdown Attack?

```c
char secret = *(char *) 0xfffffffff81a0123;
```
2018: Meltdown Attack?

```c
char secret = *(char *) 0xffffffff81a0123;
```

Virtual Address Space

- Oracle
- User Space
- Kernel Space
- Password (0xf...81a0123)

CPU Registers

- 256 different CPU Cache Line
2018: Meltdown Attack? (Step 1)

```
char secret = *(char *) 0xffffffff81a0123;
```
2018: Meltdown Attack? (Step 1)

char secret = *(char *) 0xffffffff81a0123;
2018: Meltdown Attack? (Step 2)

```c
char secret = *(char *) 0xffffffff81a0123;
char x = oracle[secret * 4096];
```
2018: Meltdown Attack? (Step 2)

```c
char secret = *(char *) 0xffffffff81a0123;
char x = oracle[secret * 4096];
```
2018: Meltdown Attack? (Step 3)

char secret = *(char *) 0xffffffff81a0123;
char x = oracle[secret * 4096];
char secret = *(char *) 0xffffffff81a0123;
char x = oracle[secret * 4096];
2018: Meltdown Attack? (Step 3)

```c
char secret = *(char *) 0xffffffff81a0123;
char x = oracle[secret * 4096];
```
2018: Meltdown Attack? (Step 3)

```c
char secret = *(char *) 0xffffffff81a0123;
char x = oracle[secret * 4096];
```
Microarchitecture Data Sampling (MDS)

- Meltdown is fixed but you can still leak on the fix hardware.
- Which part of the CPU leak the data?!

- Why does it leak?
ZombieLoad Attack

mov 0x401234, %rsi
mov (%rsi), %rax
ZombieLoad Attack

\textbf{mov} \texttt{0x401234, %rsi} \\
\textbf{mov} (\%rsi), \%rax
ZombieLoad Attack

De-allocate

Core

L1D Cache

LFB (10 entries)

Cache Line

L2

L3

DRAM
ZombieLoad Attack

Variant 1: #GP

Variant 2: #RTM (AKA TAA)

Variant 3: MC

P RW US ... A ... Physical Page Number ...
CPU Memory Subsystem - Leaky Buffers

- **MSBDS**
- **MLPDS**
- **MFBDS**

**Memory Subsystem**

- **Store Buffer**
  - DATA
  - PFN [8:0]
  - VFN
  - Offset

- **Load Buffer**
  - DATA
  - PFN
  - VFN
  - Offset

- **Fill Buffer**

- **L1**
- **L2**
- **L3**
- **DRAM**
- **DTLB**
CPU Memory Subsystem

Allocation Queue

stor $$, (add_A)
CPU Memory Subsystem

Front End

Allocation Queue
stor $$, (add_A)

Back End

ROB
Scheduler

EUs
- Store
- Load
- ALU
- ALU
CPU Memory Subsystem

Front End
- Allocation Queue
- `stor $$, (add_A)`
- Scheduler

Back End
- ROB
- EUs
  - Store
  - Load
  - ALU
- ALU

Memory Subsystem
- Store Buffer
  - DATA
  - PFN [8:0]
  - VFN
  - Offset
CPU Memory Subsystem

Front End
- Allocation Queue
  - stor $\$, (add_A)

Back End
- Scheduler
- ROB
- EUs
  - Store
  - Load
  - ALU

Memory Subsystem
- Store Buffer
  - DATA
  - PFN [8:0]
  - VFN
  - Offset
  - ...

- Fill Buffer
- DTLB

- L1
- L2
- L3
- DRAM
Store Virtual Address
0x000401

DTLB

<table>
<thead>
<tr>
<th>P</th>
<th>RW</th>
<th>US</th>
<th>...</th>
<th>A</th>
<th>...</th>
<th>Physical Page Number</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>RW</td>
<td>US</td>
<td>...</td>
<td>A</td>
<td>...</td>
<td>Physical Page Number</td>
<td>...</td>
</tr>
<tr>
<td>P</td>
<td>RW</td>
<td>US</td>
<td>...</td>
<td>A</td>
<td>...</td>
<td>Physical Page Number</td>
<td>...</td>
</tr>
<tr>
<td>P</td>
<td>RW</td>
<td>US</td>
<td>...</td>
<td>A</td>
<td>...</td>
<td>Physical Page Number</td>
<td>...</td>
</tr>
<tr>
<td>P</td>
<td>RW</td>
<td>US</td>
<td>...</td>
<td>A</td>
<td>...</td>
<td>Physical Page Number</td>
<td>...</td>
</tr>
</tbody>
</table>

Memory Subsystem

CPU Memory Subsystem

Allocation Queue
stor $s, (add_A)

Front End

Back End

EUs

Store

Load

Store Buffer

DATA | PFN [8:0] | VFN | Offset
...  | ....    | .... | ...
DATA | PFN [8:0] | VFN | Offset
...  | ....    | .... | ...
DATA | PFN [8:0] | VFN | Offset

VFN
PFN [8:0]

L1 Fill Buffer

L1

DTLB

L2

L3

DRAM

27
CPU Memory Subsystem

Store Virtual Address

0x000401

DTLB

<table>
<thead>
<tr>
<th>P</th>
<th>RW</th>
<th>US</th>
<th>...</th>
<th>A</th>
<th>...</th>
<th>Physical Page Number</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>RW</td>
<td>US</td>
<td>...</td>
<td>A</td>
<td>...</td>
<td>Physical Page Number</td>
<td>...</td>
</tr>
<tr>
<td>P</td>
<td>RW</td>
<td>US</td>
<td>...</td>
<td>A</td>
<td>...</td>
<td>Physical Page Number</td>
<td>...</td>
</tr>
<tr>
<td>P</td>
<td>RW</td>
<td>US</td>
<td>...</td>
<td>A</td>
<td>...</td>
<td>Physical Page Number</td>
<td>...</td>
</tr>
</tbody>
</table>
CPU Memory Subsystem

Store Virtual Address

0x000401

DTLB

<table>
<thead>
<tr>
<th>P</th>
<th>RW</th>
<th>US</th>
<th>...</th>
<th>A</th>
<th>...</th>
<th>Physical Page Number</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>RW</td>
<td>US</td>
<td>...</td>
<td>A</td>
<td>...</td>
<td>Physical Page Number</td>
<td>...</td>
</tr>
<tr>
<td>P</td>
<td>RW</td>
<td>US</td>
<td>...</td>
<td>A</td>
<td>...</td>
<td>Physical Page Number</td>
<td>...</td>
</tr>
</tbody>
</table>

PMH

Page Walk

Allocation Queue

stor $$, (add_A)
CPU Memory Subsystem

Front End

Allocation Queue
stor $$, (add_A)

Scheduler

Back End

ROB

EUs

Load
ALU
ALU

Store

Store Buffer
DATA | PFN [8:0] | VFN | Offset
DATA | PFN [8:0] | VFN | Offset
... | .... | ... | ...
DATA | PFN [8:0] | VFN | Offset

Memory Subsystem

L1

Fill Buffer

DTLB

L2

L3

DRAM
CPU Memory Subsystem

Front End

Allocation Queue
load (add_B), AX

Scheduler

Back End

ROB

EUs

Store
Load
ALU
ALU

Memory Subsystem

Store Buffer

<table>
<thead>
<tr>
<th>DATA</th>
<th>PFN [8:0]</th>
<th>VFN</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA</td>
<td>PFN [8:0]</td>
<td>VFN</td>
<td>Offset</td>
</tr>
<tr>
<td>...</td>
<td>....</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>DATA</td>
<td>PFN [8:0]</td>
<td>VFN</td>
<td>Offset</td>
</tr>
</tbody>
</table>

Fill Buffer

DTLB

L1

L2

L3

DRAM
CPU Memory Subsystem

Front End
- Allocation Queue
  - load (add_B), AX

Back End
- Scheduler
- ROB
- EUs
  - Store
  - Load
  - ALU

Memory Subsystem
- Load Buffer
  - DATA
  - PFN
  - VFN
  - Offset
- Store Buffer
  - DATA
  - PFN
  - VFN
  - Offset
- Fill Buffer
- DTLB
- L1
- L2
- L3
- DRAM
CPU Memory Subsystem

Front End
- Allocation Queue
- stor $$, (add_A)
- stor ##, (add_B)
- load (add_C), CX
- add CX, BX

Back End
- ROB
- Scheduler
- EUs
  - Store
  - Load
  - ALU
  - ALU

Memory Subsystem
- Store Buffer
  - DATA
  - PFN [8:0]
  - VFN
  - Offset

- Load Buffer
  - DATA
  - PFN
  - VFN
  - Offset

- DRAM
- L1
  - Fill Buffer
  - DTLB
- L2
- L3
CPU Memory Subsystem - Store Forwarding

Front End
- Allocation Queue
  - stor $$, (add_A)
  - stor $$, (add_B)
  - load (add_C), CX
  - add CX, BX

Back End
- Scheduler
- ROB
- EUs
  - Store
  - Load
  - ALU

Memory Subsystem
- DRAM
- L1
- L2
- L3
- DTLB
- Fill Buffer

Store Buffer
- DATA
- PFN [8:0]
- VFN
- Offset

Load Buffer
- DATA
- PFN
- VFN
- Offset

Load Buffer
- DATA
- PFN
- VFN
- Offset
CPU Memory Subsystem - Store Forwarding

- $\text{addr}_c = \text{addr}_a$?
- $\text{addr}_c = \text{addr}_b$?
Virtual Address

| VFN | Offset |
Virtual Address

<table>
<thead>
<tr>
<th>VFN</th>
<th>Offset</th>
</tr>
</thead>
</table>

PTE

<table>
<thead>
<tr>
<th>P</th>
<th>RW</th>
<th>US</th>
<th>...</th>
<th>A</th>
<th>...</th>
<th>Physical Page Number</th>
<th>...</th>
</tr>
</thead>
</table>
Virtual Address

<table>
<thead>
<tr>
<th>VFN</th>
<th>Offset</th>
</tr>
</thead>
</table>

PTE

| P | RW | US | ... | A | ... | Physical Page Number | ... |
Memory Access

Canonical

Y

TLB

Y

Perm.

Y

Present

#GP

PMH

#PF

Virtual Address

<table>
<thead>
<tr>
<th>VFN</th>
<th>Offset</th>
</tr>
</thead>
</table>

PTE

<table>
<thead>
<tr>
<th>P</th>
<th>RW</th>
<th>US</th>
<th>...</th>
<th>A</th>
<th>...</th>
<th>Physical Page Number</th>
<th>...</th>
</tr>
</thead>
</table>
Virtual Address

<table>
<thead>
<tr>
<th>VFN</th>
<th>Offset</th>
</tr>
</thead>
</table>

PTE

<table>
<thead>
<tr>
<th>P</th>
<th>RW</th>
<th>US</th>
<th>...</th>
<th>A</th>
<th>...</th>
<th>Physical Page Number</th>
<th>...</th>
</tr>
</thead>
</table>
Virtual Address

| VFN | Offset |

PTE

| P | RW | US | ... | A | ... | Physical Page Number | ... |
Memory Access

Canonical Y TLB Y Perm. Y Present Y Accessed

#GP PMH #PF Y

 Perm. Present Y Accessed

#PF

Perm. Present Y Accessed

#GP

Virtual Address

| VFN | Offset |

PTE

| P | RW | US | ... | A | ... | Physical Page Number | ... |
Memory Access

Canonical → TLB → Perm. → Present → Accessed

#GP → PMH

Perm. → Present → Accessed

Present → #PF

Accessed → Set A Bit

Cached → Cache Aligned → Aligned Vector

Cached → Cache Miss Handler

Cache Aligned → Split Cache

Aligned Vector → #GP

Virtual Address

<table>
<thead>
<tr>
<th>VFN</th>
<th>Offset</th>
</tr>
</thead>
</table>

PTE

<table>
<thead>
<tr>
<th>P</th>
<th>RW</th>
<th>US</th>
<th>...</th>
<th>A</th>
<th>...</th>
<th>Physical Page Number</th>
<th>...</th>
</tr>
</thead>
</table>
Virtual Address

<table>
<thead>
<tr>
<th>VFN</th>
<th>Offset</th>
</tr>
</thead>
</table>

PTE

<table>
<thead>
<tr>
<th>P</th>
<th>RW</th>
<th>US</th>
<th>...</th>
<th>A</th>
<th>...</th>
<th>Physical Page Number</th>
<th>...</th>
</tr>
</thead>
</table>
Memory Access

Canonical Y
 TLB Y
 Perm. Y
 Present Y
 Accessed

Set A Bit Y

TSX Failure Y
 False Store Dep. Y
 Hazard Recovery

Cached Y
 Cache Miss Handler

Cache Aligned Y
 Split Cache

Aligned Vector

#GP
 PMH

#PF

#RTM

Virtual Address

<table>
<thead>
<tr>
<th>VFN</th>
<th>Offset</th>
</tr>
</thead>
</table>

PTE

<table>
<thead>
<tr>
<th>P</th>
<th>RW</th>
<th>US</th>
<th>...</th>
<th>A</th>
<th>...</th>
<th>Physical Page Number</th>
<th>...</th>
</tr>
</thead>
</table>
CPU Memory Subsystem - Hazard Recovery

Front End
- Allocation Queue
  - stor $$, (addr_B)
  - load (addr_A), AX

Back End
- ROB
- Scheduler
- EUs
  - Store
  - Load
  - ALU

Memory Subsystem
- L1
  - Fill Buffer
  - DTLB
- L2
- L3
- DRAM
CPU Memory Subsystem - Hazard Recovery

Front End
- Allocation Queue
  - stor $$, (addr_B)
  - load (addr_A), AX

Back End
- ROB
- EUs
  - Store
  - Load
- Scheduler
- ALU

Back End
- Memory Subsystem
  - L1
  - L2
  - L3
  - DRAM
  - DTLB
  - Fill Buffer

Store Buffer
- DATA
- PFN [8:0]
- VFN
- Offset
- ...
- ...
- ...
- ...

Load Buffer
- DATA
- PFN
- VFN
- Offset
- ...
- ...
- ...
- ...

Load
- ALU
CPU Memory Subsystem - Hazard Recovery

Front End
- Allocation Queue
  - stor $$, (addr_B)
  - load (addr_A), AX

Back End
- Scheduler
- ROB
  - Load
  - ALU
- EUs
  - Store
  - Load
  - ALU

Memory Subsystem
- Store Buffer
  - DATA
  - PFN [8:0]
  - VFN
  - Offset
- Load Buffer
  - DATA
  - PFN
  - VFN
  - Offset
- Fill Buffer
- DTLB
- L1
- L2
- L3
- DRAM
CPU Memory Subsystem - Hazard Recovery

Front End
- Allocation Queue
  - stor $$, (addr_B)
  - load (addr_A), AX

Back End
- ROB
- Scheduler
- EUs
  - Store
  - Load
  - ALU

Memory Subsystem
- DRAM
- L3
- L2
- L1
- Fill Buffer
- DTLB
- Load Buffer
- Store Buffer

DATA PFN [8:0] VFN Offset
DATA PFN [8:0] VFN Offset
... ... ... ...
DATA PFN [8:0] VFN Offset
DATA PFN VFN Offset
... ... ... ...
DATA PFN VFN Offset
... ... ... ...
DATA PFN VFN Offset
... ... ... ...
DATA PFN VFN Offset
Challenges with MDS Testing?

• Reproducing attacks is not reliable. It may depend on:
  • massaging the pipeline with other instructions
  • CPU configuration (generation, frequency, microcode patch and etc)

• No public tool to find new variants or to verify hardware patches:
  • Too many things to test (Addressing mode, cache state, assists, and faults)
  • Previous POCs may not work after MC update, but what does it mean?

• Impossible to quantify the impact of leakage:
  • We should care about leakage rate and what data is leaked.
  • My POC is faster than your POC!!
Let’s see this problem in action?! (Demo)
Transynther
Transynther (Fuzzing-based Random MDS Testing)

Step 1:

```c
char secret = *(char *) 0xfffffffff81a0123;
```

Step 2:

```c
char x = oracle[secret * 4096];
```

Step 3:

256 different CPU Cache Line

'P' = 0x50
Transynther (Fuzzing-based Random MDS Testing)

Step 1:
Step 2:
Step 3:

```python
char x = oracle[secret * 4096];
```

256 different CPU Cache Line

'P' = 0x50
Transynther (Fuzzing-based Random MDS Testing)

**Step 0: Buffer Grooming**

**Step 1:**

**Step 2:**

**Step 3:**

```c
char x = oracle[secret * 4096];
```

‘T’ = 0x50

256 different CPU Cache Line
Transynther (Fuzzing-based Random MDS Testing)

Step 0: Buffer Grooming

- Stores Same Thread: 0x41424344
- Loads Same Thread: 0x51525354
- Stores Hyper Thread: 0x61626364
- Loads Hyper thread: 0x71727374

Step 1:

- Canonical
- TLB
- Aligned Vector
- Cached
- Cache Aligned
- Perm.
- Present
- False Store Dep.
- Accessed
- TSX Failure

Step 2:

\[ \text{char } x = \text{oracle[secret } \times 4096]; \]

Step 3:

\[ \text{'; } = 0x50 \]

256 different CPU Cache Line
Transynther (Fuzzing-based MDS Testing)
Transynther (Fuzzing-based MDS Testing)
Transynther (Fuzzing-based MDS Testing)
<table>
<thead>
<tr>
<th>Case</th>
<th>Preparation</th>
<th>Store</th>
<th>Load</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(access $\emptyset$, random instructions)</td>
<td>-</td>
<td>$\leftarrow + \emptyset / \emptyset / \emptyset$</td>
<td>MLPDS</td>
</tr>
<tr>
<td>2</td>
<td>(access $\emptyset$, random instructions)</td>
<td>-</td>
<td>AVX $\leftarrow + \emptyset / \emptyset / \emptyset / \emptyset$</td>
<td>MLPDS</td>
</tr>
<tr>
<td>3</td>
<td>(access $\emptyset$, random instructions)</td>
<td>-</td>
<td>AVX $\leftrightarrow + \emptyset / \emptyset / \emptyset / \emptyset$</td>
<td>Medusa</td>
</tr>
<tr>
<td>4</td>
<td>(access $\emptyset$, random instructions)</td>
<td>-</td>
<td>AVX $\leftrightarrow + \emptyset / \emptyset / \emptyset / \emptyset$</td>
<td>Medusa</td>
</tr>
<tr>
<td>5</td>
<td>-</td>
<td>store (to load)</td>
<td>$\emptyset / \emptyset / \emptyset / \emptyset / \checkmark$</td>
<td>S2L</td>
</tr>
<tr>
<td>6</td>
<td>(rep mov + store, store + fence + load)</td>
<td>store (to load)</td>
<td>$\emptyset / \emptyset / \emptyset / \checkmark$</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>-</td>
<td>store (4K Aliasing) + $\emptyset / \emptyset / \emptyset / \checkmark$</td>
<td>$\emptyset / \emptyset / \emptyset / \checkmark$</td>
<td>MSBDS, S2L</td>
</tr>
<tr>
<td>8</td>
<td>-</td>
<td>store (4K Aliasing, to load) + $\emptyset / \emptyset / \emptyset / \checkmark$</td>
<td>$\emptyset / \emptyset / \checkmark$</td>
<td>MSBDS</td>
</tr>
<tr>
<td>9</td>
<td>(Sibling on/off)</td>
<td>store (random address) + $\emptyset$</td>
<td>$\checkmark$</td>
<td>MSBDS</td>
</tr>
<tr>
<td>10</td>
<td>(Sibling on/off + clflush (store address))</td>
<td>store (Cache Offset of Load) + $\emptyset$</td>
<td>$\checkmark$</td>
<td>MSBDS</td>
</tr>
<tr>
<td>11</td>
<td>(Sibling on/off + repmov (to Load))</td>
<td>store (to Load)</td>
<td>$\emptyset / \emptyset / \emptyset / \checkmark$</td>
<td>Medusa, MLPDS</td>
</tr>
<tr>
<td>12</td>
<td>-</td>
<td>Store (Unaligned to Load)</td>
<td>AVX $\leftrightarrow + \emptyset / \emptyset / \emptyset / \checkmark$</td>
<td>Medusa</td>
</tr>
<tr>
<td>13</td>
<td>(random instructions)</td>
<td>AVX Store (to Load)</td>
<td>$\emptyset / \emptyset / \checkmark$</td>
<td>MSBDS</td>
</tr>
<tr>
<td>14</td>
<td>-</td>
<td>random fill stores</td>
<td>$\checkmark$</td>
<td>MSBDS</td>
</tr>
</tbody>
</table>

$\checkmark$ Non-canonical Address Fault  $\emptyset$ Non-present Page Fault  $\emptyset$ Supervisor Protection Fault  $\leftrightarrow$ AVX Alignment Fault
$\emptyset$ Access-bit Assist  $\leftarrow$ Split-Cache Access Assist  $\checkmark$ Access without fault or Assist
Demo some Interesting Leakage Pattern
MDS Attacks (ZombieLoad, RIDL, Fallout, ...)

• The CPU must flush the pipeline before executing an assist.

• Upon an Exception/Fault/Assist on a Load, Intel CPUs:
  • Execute the load until the last stage.
  • Flush the pipeline at the retirement stage (Cheap Recovery Logic).
  • Continue the pipeline at the retirement stage.

• Which data? (Fill buffer, Store Buffer, Load Buffer)
• Which one will be leaked first? (First come first serve)
Medusa Attack

- Write Combining fills up the entire Data Bus.
- Medusa only leaks the Upper-half of the Data Bus.
- Implicit WC, i.e., ‘rep mov’, ‘rep stos’, can be leaked.
- Served by a Write Combining Buffer (or just the the Fill Buffer)

- Advantages:
  - Prefiltered data
  - Less Noise
  - More targeted (maybe also a disadvantage)
An invalid (Non-canon) address:
0x5550000000000008-20
Medusa Attack - V1 Cache Indexing

Cache Line Index

| 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte |

An invalid (Non-canon) address:
0x5550000000000008-20
### Medusa Attack - V1 Cache Indexing

#### Cache Line Index

<table>
<thead>
<tr>
<th>8-byte</th>
<th>8-byte</th>
<th>8-byte</th>
<th>8-byte</th>
<th>8-byte</th>
<th>8-byte</th>
<th>8-byte</th>
<th>8-byte</th>
</tr>
</thead>
</table>

An invalid (Non-canon) address:
0x5550000000000008-20

---

**Faulty Load**

---

Image of a chart showing leaked cache lines vs cache-line offset.
Medusa Attack - V1 Cache Indexing

Cache Line Index

| 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte |

Common Data Bus?!
Medusa Attack - V2 Unaligned S2L Forwarding

<table>
<thead>
<tr>
<th>Cache Line Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-byte</td>
</tr>
</tbody>
</table>

Faulty Load
Medusa Attack - V2 Unaligned S2L Forwarding

Cache Line Index

Faulty Load

YMMx

REPMOV on the Hyper thread:

ABCDEFGH IJKLMNOP QRSTUVWX YZ...
Medusa Attack - V2 Unaligned S2L Forwarding

Cache Line Index

Store

Faulty Load

YMMx

REPMOV on the Hyper thread:

ABCDEFGH IJKMNOP QRSTUVWX YZ...
Medusa Attack - V2 Unaligned S2L Forwarding

Cache Line Index

| 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte |

Store

Faulty Load

YMMx

REPMOV on the Hyper thread:

ABCDEFGH IJKLMNOP QRSTUVWX YZ...
Medusa Attack - V2 Unaligned S2L Forwarding

**Cache Line Index**

| 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte |

**Faulty Load**

**YMMx**

**Store**

REPMOV on the Hyper thread:

ABCDEFGH IJKLMNOPQRSTUVWXYZ...
Medusa Attack - V2 Unaligned S2L Forwarding

Cache Line Index

| 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte |

Store

Faulty Load
Medusa Attack - V3 Shadow *REP MOV*

- A *REP MOV* that fault on the load leaks:
  - the data from the legitimate store address
  - but also the data from the *REP MOV* running on the hyper thread

```
HT 1: REP MOV
Valid Store, Faulty Load

AAAAAAAAAAAAAAA
AAAAAAAAAAAAAAA
```

```
HT 1: REP MOV
Valid Store, Faulty Load

ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

MD Leak
Medusa Attack - V3 Shadow \textit{REP MOV}

- A \textit{REP MOV} that fault on the load leaks:
  - the data from the legitimate store address
  - but also the data from the \textit{REP MOV} running on the hyper thread

\begin{itemize}
\item HT 1: REP MOV
  \begin{itemize}
  \item Valid Store, Faulty Load
  \end{itemize}
\end{itemize}

\begin{itemize}
\item MD Leak
\end{itemize}

\begin{itemize}
\item AAAAAAAABBBBBBBB
  \item AAAAAAAAAAAAAAA
\end{itemize}

\begin{itemize}
\item ABCDEFGHIJKLMNOP
  \item AAAAAAAAAAAAAAAA
\end{itemize}

\begin{itemize}
\item AAAAAAAAAIIIIIIIIIAAAAAIIIIIIIIIIIIIIIIIIIIIIIIIAAAAAAA...
\end{itemize}
OpenSSL RSA Key Recovery

• OpenSSL Base64 Decoder uses inline `Memcpy(-oS)
• Triggerred during the RSA Key Decoding from the PEM format:

-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQDmTvQjtitGtnIqMwmmalW+YjbyTsNR8PGKXr78iYwrMV5Ye4VGy
BwS6qLD4s/EzCzGIDwkWVCVx+gVHvh2wGW15Ddf0f0VAAtAMkR6gRABy4TkK+6YFSK
AyjmHvKcfFHvc9loeFGDymwFFkfdwzppXnH1Wwt0lnyCU1GbQ1w7AHuwIDAQAB
AoGBAMyDriT7pQ29NBlfMmGQuFtw8c0R3EamlIdQbX7qUguFEoe2YHqjdrKho5oZj
nDu8o+Zzm5jzBSzd7oZ4qaeekv0fO+ZSz6CKYLbuzG2IXUB8nHJ7NulH3lacfivD
V4CfgoYFnTK+MDG/xTVqywrCTsslTCYC/XZOXU5Xt5z32FZAkEA/nLWQhMC4YPM
0LqmMtgKzfgQdJ7vbr43WVVNpC/dN/ibUASI/3YwY0uUtqSjIlgly7pRohrPJ6W
ntSjw0UAhQJBAOe2b9cfiOTFKXxyU4j315VkulFFTyL6GwXi/7mvpCDixDLRNryk
uRigmdKjtIUrAX0pwjgxA6niqJ691jExez8CQQCccMZZAvTbZHsn9LwhxqS0SIY1
K+ZxX5ogirFDPS5NQzyE7adSntSioh6/LQKBX6BAR9FwtxBPActzw5F9geZAkA8
a3z0SlvG04aC1cjkUGPxs6wxxb79F2RhmsKRbvh7JiYk3RQ+L7vJgmWPGu5AcLM
oVPsJmbbkKfJZTynVOW/AkABepEi++ZQQW0FXJWZ3nM+2CNcXYCtTgi4bGkvnZPp/
1pAy9rzeVYhba8acTRnt+dU+UZ74CTfuzUTZLOluVe
-----END RSA PRIVATE KEY-----
OpenSSL RSA Key Recovery

- OpenSSL Base64 Decoder uses inline Memcpy(-oS)
- Triggered during the RSA Key Decoding from the PEM format:

```
-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQDmTvQjJTGtlnqMwmmMLw+YjbyTsNJR8PGKXR78iYwMrV5Ye4VGyBwS6qLD4/s/EzCzGIDwkWCVx+gVHvh2wGW15Ddof0gVAtAMkR6gRABy4TkK+6YFSK
AyjmHvKcfFHxc9loeFDyjmwFFkdfwzppXnH1Wyt0OlnyCU1GbQ1w7AHuwIDAQAB
AoGBAMyDri7pQ29NBlfMmGQuFtw8c0R3EamldQbX7qUguFEoe2YHqjdrKho5oZj
nDu8o+Zzm5jzBSzdf70Z4qaeekv0fO+ZSz6CKYLbuzG2IXUB8hJ7NuH3lacfivD
V4Cfg0yFnTK+MDG/xTVqywrCTsskTCYCY/ZOXUX5X5z32FZAKEA/nLWQhMC4YPM
0LqMtqKzfgQdJ7vbr43WVVPNpCd/n/ibUASI/3YwY0uUtqSjllghlY7pRohrPJ6W
nt5Jw0UAhQJBAOe2b9cfiQTFKxyU4j315VkulFFyl6GwXi/7mvpcDCixDLNRyk
uRigmdKjtIUrAX0pwjgXa6niqJ691jExez8CQQCcMZZAvTbZhhSN9LwHxqSOSIY1
K+ZxX5ogirFDP5NQzyE7adSntSiohL/LQKBX6BAR9FwtxBPACtwz5F9geZAkA8
a3z0S1vG04aC1ckjgUPsx6wxbl79F2RhmSRhv7jiYK3RQ+L7vJgmpWPGr5AcLM
oVPsJmbbkKfJZNTyVOW/AkABepEi++ZQQW0FXJWZ3nM+2CNcXYCtTgi4bGkvnZPp
/1pAy9rjeVJYhb8acTRnt+dU+U6Z4CTfuzUTZLOluVe
-----END RSA PRIVATE KEY-----
```
OpenSSL RSA Key Recovery

- OpenSSL Base64 Decoder uses inline Memcpy(-oS)
- Triggerred during the RSA Key Decoding from the PEM format:

<table>
<thead>
<tr>
<th>N (Modulus)</th>
</tr>
</thead>
<tbody>
<tr>
<td>d (Private Key)</td>
</tr>
<tr>
<td>P</td>
</tr>
<tr>
<td>d mod (p-1)</td>
</tr>
<tr>
<td>d mod (q-1)</td>
</tr>
<tr>
<td>Q^(-1) mod p</td>
</tr>
</tbody>
</table>
OpenSSL RSA Key Recovery - Coppersmith

- Knowledge of at least $\frac{1}{3}$ of $P+Q$
- Create a $n$ dimensional hidden number problem where $n$ is relative to the number of recovered chunks
- Feed it to the lattice-based algorithm to find the short vector $PQ$
OpenSSL RSA Key Recovery - Coppersmith Attack

- Knowledge of at least $\frac{1}{3}$ of $P+Q$.
- Creating a $n$ dimensional hidden number problem where $n$ is relative to the number of recovered chunks.
- Feeding it to the lattice-based algorithm to find the short vector.
Conclusion

- Automated Testing for CPU Attacks,
  - helps us to understand the root cause of these issues better.
  - can be used to verify hardware mitigations (e.g., Fallout on ICL).
  - can help us to improve the leakage rate and understand the impact of attacks better.

- The impact of attacks depend also on the exploitation technique.

- Potentials and Future work:
  - Can we integrate such tools with feedback from hardware/simulator?
CPU Memory Subsystem

Front End
- Allocation Queue
- stor $$, (add_A)
- stor ##, (add_B)
- load (add_C), CX
- add CX, BX

Back End
- Scheduler
- ROB
- EUs
- Store
- Load
- ALU
- ALU

Memory Subsystem
- EUs
- Store Buffer
- Load Buffer
- Fill Buffer
- DTLB
- L1
- L2
- L3
- DRAM