Microarchitectural Attacks: From the Basics to Arbitrary Read and Write Primitives without any Software Bugs

Daniel Gruss
April 11, 2018
Graz University of Technology
• Daniel Gruss
• Post-Doc @ Graz University of Technology
• Twitter: @lavados
• Email: daniel.gruss@iaik.tugraz.at
printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);
CPU Cache

printf("%d", i);  
printf("%d", i);

Cache miss

Request
Response

i

Daniel Gruss — Graz University of Technology
printf("%d", i);
printf("%d", i);

Cache miss

Cache hit

Request
Response

Daniel Gruss — Graz University of Technology
CPU Cache

printf("%d", i);

Cache miss

DRAM access,
slow

Request

Response

printf("%d", i);

Cache hit

i

DRAM access,
slow

printf("%d", i);

Cache miss

i

Cache hit

Request

Response
CPU Cache

printf("%d", i);
printf("%d", i);

Cache miss
Cache hit

No DRAM access, much faster
DRAM access, slow

Request
Response

Daniel Gruss — Graz University of Technology
Flush+Reload

Shared Memory

ATTACKER
flush
access

VICTIM
access
Flush+Reload

ATTACKER

flush
access

Shared Memory

cached

VICTIM

cached
access

Daniel Gruss — Graz University of Technology
Flush+Reload

ATTACKER

flush
access

Shared Memory

VICTIM

access

Daniel Gruss — Graz University of Technology
Flush+Reload

ATTACKER
flush
access

Shared Memory

VICTIM
access
FlushReload

ATTACKER

Shared Memory

VICTIM

flush

access

access
Flush+Reload

ATTACKER

flush
access

Shared Memory

VICTIM

access
Flush+Reload

ATTACKER

flush
access

Shared Memory

VICTIM

access

Daniel Gruss — Graz University of Technology
Flush+Reload

ATTACKER

flush
access

FAST if victim accessed data,
SLOW otherwise

Shared Memory

VICTIM

access

Daniel Gruss — Graz University of Technology
Memory Access Latency

![Graph showing Memory Access Latency]

- **X-axis (Latency in Cycles)**: The range is from 0 to 500 cycles.
- **Y-axis (Number of Accesses)**: The range is from 1 to 10,000,000 accesses.
- **Legend**:
  -Cached
  -Not Cached

The graph illustrates the latency in cycles for both cached and not cached access patterns.
int width = 10, height = 5;

float diagonal = sqrt(width * width + height * height);

int area = width * height;

printf("Area %d x %d = %d\n", width, height, area);
int width = 10, height = 5;

float diagonal = sqrt(width * width
   + height * height);

int area = width * height;

printf("Area %d x %d = %d\n", width, height, area);
Let’s see whether this works...

Adapted code

```c
1  *(volatile char*)0;
2  array[0] = 0;     // will be cached
```
Let’s hope this does not work...

- Maybe there is no permission check in transient instructions...
Let’s hope this does not work...

- Maybe there is no permission check in transient instructions...
- ...or it is only done when committing them

\[
1. \text{char } \text{data} = * (\text{char}*) 0 \text{xffffffff81a000e0}
\]

\[
2. \text{array} [\text{data}*4 0 9 6] = 0
\]
Let’s hope this does not work...

- Maybe there is no permission check in transient instructions...
- ...or it is only done when committing them
- Indirection through microarchitectural traces:

```
char data = *(char*)0xffffffff81a000e0;
array[data * 4096] = 0;
```
Let's hope this does not work...

- Maybe there is no permission check in transient instructions...
- ...or it is only done when committing them
- Indirection through microarchitectural traces:

```c
char data = *(char*)0xffffffff81a000e0;
array[data * 4096] = 0;
```

- Check whether any part of array is cached
Flush+Reload over all pages of the array

Index of cache hit reveals data
Flush+Reload over all pages of the array

Index of cache hit reveals data

Permission check is in some cases not fast enough
Spying on passwords

```
mschwarz@lab06:~/Documents$
```
Leaking a picture like in CSI Cyber

meltdown@meltdown ~/ppm2 % taskset 1 ./imgdump 0x375a00000 14919 > output.flif
Reading from 0xffff880375a00000
Leaking Passwords from your Password Manager

Daniel Gruss — Graz University of Technology
DRAM organization

channel 0

channel 1
DRAM organization

channel 0

back of DIMM: rank 1

front of DIMM: rank 0

channel 1

rank 0

rank 1
DRAM organization

channel 0

back of DIMM: rank 1

front of DIMM: rank 0

channel 1

chip

Daniel Gruss — Graz University of Technology
DRAM organization

Chip

Bank 0

- Row 0
- Row 1
- Row 2
- ... (continued)
- Row 32767

Row buffer

64k cells
1 capacitor, 1 transistor each
Rowhammer

- Cells leak → repetitive refresh necessary
- Maximum interval between refreshes to guarantee data integrity
- Cells leak faster upon proximate accesses → Rowhammer

---

Daniel Gruss — Graz University of Technology
Rowhammer

- Cells leak $\rightarrow$ repetitive refresh necessary
- Maximum interval between refreshes to guarantee data integrity
- Cells leak faster upon proximate accesses $\rightarrow$ Rowhammer
Rowhammer

- Cells leak → repetitive refresh necessary
- Maximum interval between refreshes to guarantee data integrity
- Cells leak faster upon proximate accesses → Rowhammer
Rowhammer

- Cells leak → repetitive refresh necessary
- Maximum interval between refreshes to guarantee data integrity
- Cells leak faster upon proximate accesses → Rowhammer
- Cells leak $\rightarrow$ repetitive refresh necessary
- Maximum interval between refreshes to guarantee data integrity
- Cells leak faster upon proximate accesses $\rightarrow$ Rowhammer
Rowhammer

- Cells leak → repetitive refresh necessary
- Maximum interval between refreshes to guarantee data integrity
- Cells leak faster upon proximate accesses → Rowhammer
#1 - Single-sided hammering

DRAM bank

activate

11111111111111
11111111111111
11111111111111
11111111111111
11111111111111
11111111111111
11111111111111
11111111111111
11111111111111
#1 - Single-sided hammering

DRAM bank

activate
# Single-sided hammering

**DRAM bank**

<table>
<thead>
<tr>
<th>11111111111111111111</th>
<th>11111111111111111111</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111111111111111111</td>
<td></td>
</tr>
<tr>
<td>11111111111111111111</td>
<td></td>
</tr>
<tr>
<td>11111111111111111111</td>
<td></td>
</tr>
<tr>
<td>11111111111111111111</td>
<td></td>
</tr>
<tr>
<td>11111111111111111111</td>
<td></td>
</tr>
<tr>
<td>11111111111111111111</td>
<td></td>
</tr>
<tr>
<td>11111111111111111111</td>
<td></td>
</tr>
<tr>
<td>11111111111111111111</td>
<td></td>
</tr>
<tr>
<td>11111111111111111111</td>
<td></td>
</tr>
<tr>
<td>11111111111111111111</td>
<td></td>
</tr>
<tr>
<td>11111111111111111111</td>
<td></td>
</tr>
<tr>
<td>11111111111111111111</td>
<td></td>
</tr>
</tbody>
</table>

activate
#1 - Single-sided hammering

DRAM bank

activate
#1 - Single-sided hammering

DRAM bank

activate
#1 - Single-sided hammering

DRAM bank

111111111111111
111111111111111
10111110101111
111111111111111
111111111111111
111111111111111
111111111111111
111111111111111
111111111111111
111111111111111

bit flips
#2 - Double-sided hammering

- **DRAM bank**

```
activate

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Daniel Gruss — Graz University of Technology
#2 - Double-sided hammering

DRAM bank

1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1

activate
Double-sided hammering

DRAM bank

activate

1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1
#2 - Double-sided hammering

**DRAM bank**

```
activate
```

```
111111111111111
111111111111111
111111111111111
111111111111111
```

```
111111111111111
111111111111111
111111111111111
111111111111111
```

```
111111111111111
111111111111111
111111111111111
111111111111111
```

```
111111111111111
111111111111111
111111111111111
111111111111111
```

```
111111111111111
111111111111111
111111111111111
111111111111111
```
#2 - Double-sided hammering

**DRAM bank**

![Diagram of DRAM bank with binary values and an 'activate' label pointing to a specific row.]}
#2 - Double-sided hammering

DRAM bank

```
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 0 1 1 1 1 0 1 0 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 1 0 1 0 1 1 1 1 1 1 1 1 1
```

activate

bit flips
#3 - One-location hammering

DRAM bank

activate

1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
#3 - One-location hammering

DRAM bank

```
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Daniel Gruss — Graz University of Technology
#3 - One-location hammering

![DRAM bank diagram](image.png)

<table>
<thead>
<tr>
<th>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</th>
<th>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
</tbody>
</table>

activate

21

Daniel Gruss — Graz University of Technology
#3 - One-location hammering

DRAM bank

```
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Daniel Gruss — Graz University of Technology
#3 - One-location hammering

DRAM bank

activate

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
#3 - One-location hammering

DRAM bank

```
1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
1 0 1 1 1 1 0 1 0 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
```

bit flips
How to exploit random bit flips?

- They are not random → highly reproducible flip pattern!
  1. Choose a (kernel) data structure that you can place at arbitrary memory locations
  2. Scan for “good” flips
  3. Place (kernel) data structure there
  4. Trigger bit flip again
What if we cannot target kernel pages?

- Many applications perform actions as root
What if we cannot target kernel pages?

• Many applications perform actions as root
• They can be used by unprivileged users as well
What if we cannot target kernel pages?

- Many applications perform actions as root
- They can be used by unprivileged users as well
- sudo
Opcode Flipping - Conditional Jump

JE

0 1 1 1 0 1 0 0

→

HLT

1 1 1 1 0 1 0 0
Opcode Flipping - Conditional Jump

JE

| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |

→

XORB

| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
Opcode Flipping - Conditional Jump

JE

PUSHQ
Opcode Flipping - Conditional Jump

JE

0 1 1 1 0 1 0 0

<prefix>

0 1 1 0 0 1 0 0
Opcode Flipping - Conditional Jump

JE

0 1 1 1 0 1 0 0

JL

0 1 1 1 1 1 0 0
Opcode Flipping - Conditional Jump

JE

\[01110100\] → \[011110000\]

JO
Opcode Flipping - Conditional Jump

JE

JBE

0 1 1 1 0 1 0 0

0 1 1 1 0 1 1 0
Opcode Flipping - Conditional Jump

JE

\[ \begin{array}{cccccc}
0 & 1 & 1 & 1 & 0 & 1 \end{array} \]

JNE

\[ \begin{array}{cccccc}
0 & 1 & 1 & 1 & 0 & 1 \end{array} \]

Daniel Gruss — Graz University of Technology
We have ignored microarchitectural attacks for many many years:
We have ignored microarchitectural attacks for many many years:

- attacks on crypto
We have ignored microarchitectural attacks for many many years:

- attacks on crypto → “software should be fixed”
We have ignored microarchitectural attacks for many many years:

- attacks on crypto → “software should be fixed”
- attacks on ASLR
What do we learn from it?

We have ignored microarchitectural attacks for many many years:

- attacks on crypto → “software should be fixed”
- attacks on ASLR → “ASLR is broken anyway”
What do we learn from it?

We have ignored microarchitectural attacks for many many years:

- attacks on crypto → “software should be fixed”
- attacks on ASLR → “ASLR is broken anyway”
- attacks on SGX and TrustZone
We have ignored microarchitectural attacks for many many years:

- attacks on crypto → “software should be fixed”
- attacks on ASLR → “ASLR is broken anyway”
- attacks on SGX and TrustZone → “not part of the threat model”
We have ignored microarchitectural attacks for many many years:

- attacks on crypto → “software should be fixed”
- attacks on ASLR → “ASLR is broken anyway”
- attacks on SGX and TrustZone → “not part of the threat model”
- Rowhammer attacks
We have ignored microarchitectural attacks for many many years:

- attacks on crypto → “software should be fixed”
- attacks on ASLR → “ASLR is broken anyway”
- attacks on SGX and TrustZone → “not part of the threat model”
- Rowhammer attacks → “only affects cheap sub-standard modules”
We have ignored microarchitectural attacks for many many years:

- attacks on crypto → “software should be fixed”
- attacks on ASLR → “ASLR is broken anyway”
- attacks on SGX and TrustZone → “not part of the threat model”
- Rowhammer attacks → “only affects cheap sub-standard modules”

→ for years we solely optimized for performance
When you read the manuals...

After learning about a side channel you realize:

- the side channels were documented in the Intel manual
- only now we understand the implications
When you read the manuals...

After learning about a side channel you realize:

- the side channels were documented in the Intel manual
When you read the manuals...

After learning about a side channel you realize:

- the side channels were documented in the Intel manual
- only now we understand the implications
Attacks vs. Defenses

• moral obligation to invest more time on defenses than on attacks
• dangerous: we overlooked Meltdown and Spectre for decades
• we don’t know all problems. do we know at least the most important subset?
• are we hammering on a small subset of problems and forgot about the bigger picture?
• moral obligation to invest more time on defenses than on attacks
• moral obligation to invest more time on defenses than on attacks
• *dangerous*: we overlooked Meltdown and Spectre for decades
• moral obligation to invest more time on defenses than on attacks
• **dangerous**: we overlooked Meltdown and Spectre for decades
• we don’t know all problems. do we know at least the most important subset?
• moral obligation to invest more time on defenses than on attacks
• dangerous: we overlooked Meltdown and Spectre for decades
• we don’t know all problems. do we know at least the most important subset?
• are we hammering on a small subset of problems and forgot about the bigger picture?
Attacks vs. Defenses

www.tugraz.at

• moral obligation to invest more time on defenses than on attacks

• dangerous: we overlooked Meltdown and Spectre for decades

• we don't know all problems. do we know at least the most important subset?

• are we hammering on a small subset of problems and forgot about the bigger picture?

28

Daniel Gruss — Graz University of Technology
What do we learn from it?

A unique chance to

- rethink processor design
What do we learn from it?

A unique chance to

- rethink processor design
- grow up, like other fields (car industry, construction industry)
A unique chance to

- rethink processor design
- grow up, like other fields (car industry, construction industry)
- find good trade-offs between security and performance
A unique chance to

- rethink processor design
- grow up, like other fields (car industry, construction industry)
- find good trade-offs between security and performance
- dedicate more time into identifying problems and not solely in mitigating known problems
Microarchitectural Attacks: From the Basics to Arbitrary Read and Write Primitives without any Software Bugs

Daniel Gruss
April 11, 2018
Graz University of Technology