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

Cache miss
Request
printf("%d", i);
printf("%d", i);

Cache miss
Request
Response

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

DRAM access, slow

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

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

DRAM access,
slow

DRAM access,
slow

printf("%d", i);

printf("%d", i);

Cache hit

Request

Response

Cache miss
CPU Cache

- **DRAM access, slow**
- **Cache miss**
- **Cache hit**
- **No DRAM access, much faster**

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

Request
Response

Daniel Gruss — Graz University of Technology
Flush+Reload

Shared Memory

ATTACKER

VICTIM

flush
access

flush
access
Flush+Reload

ATTACKER

flush
access

cached

Shared Memory

VICTIM

access

cached

cached
Flush+Reload

ATTACKER

Flush
Access

Shared Memory

VICTIM

Access

[Diagram showing ATTACKER flushing and accessing shared memory, and VICTIM accessing the same shared memory]
Flush+Reload

ATTACKER

flush
access

Shared Memory

VICTIM

access
Flush+Reload

ATTACKER

flush
access

Shared Memory

access

VICTIM
Flush + Reload

ATTACKER

Shared Memory

VICTIM

flush
access

flush
access

Shared Memory
Flush+Reload

ATTACKER

flush
access

Shared Memory

fast if victim accessed data, slow otherwise

VICTIM

access

Daniel Gruss — Graz University of Technology
Memory Access Latency

Latency in Cycles

Number of Accesses

Cached
Not Cached
Cache Template Attack Demo

```bash
% sleep 2; ./spy 300 7f05140a4088-7f051417b000 r-xp 0x20000 08:02 26 8050
/usr/lib/x86_64-linux-gnu/gedit/libgedit.so
```

```
shark% ./spy
```
| Key  | g   | h   | i   | j   | k   | l   | m   | n   | o   | p   | q   | r   | s   | t   | u   | v   | w   | x   | y   | z   |
|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| Address | 0x7c680 | 0x7c6c0 | 0x7c700 | 0x7c740 | 0x7c780 | 0x7c7c0 | 0x7c800 | 0x7c840 | 0x7c880 | 0x7c8c0 | 0x7c900 | 0x7c940 | 0x7c980 | 0x7c9c0 | 0x7ca00 | 0x7cb80 | 0x7cc40 | 0x7cc80 | 0x7ccc0 | 0x7cd00 |
7. Serve with cooked and peeled potatoes
Wait for an hour
Wait for an hour

LATENCY
1. Wash and cut vegetables

2. Pick the basil leaves and set aside

3. Heat 2 tablespoons of oil in a pan

4. Fry vegetables until golden and softened
1. Wash and cut vegetables

2. Pick the basil leaves and set aside

3. Heat 2 tablespoons of oil in a pan

4. Fry vegetables until golden and softened
```c
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);
```
```c
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);
```
```
char data = *(char*)0xfffffffff81a000e0;
printf("%c\n", data);
```
```c
char data = *(char*)0xffffffff81a000e0;
printf(%c\n, data);
```

```
segfault at ffffffff81a000e0 ip 0000000000400535
sp 00007ffce4a80610 error 5 in reader
```
Building Meltdown

char data = *(char*)0xffffffff81a000e0;
printf("%c\n", data);

segfault at fffffffff81a000e0 ip 00000000000400535
sp 00007ffce4a80610 error 5 in reader

- Kernel addresses are not accessible

Daniel Gruss — Graz University of Technology
Building Meltdown

```c
char data = *(char*)0xffffffff81a000e0;
printf("%c\n", data);
```

• Kernel addresses are not accessible

• Are privilege checks also done when executing instructions out of order?

segfault at ffffffff81a000e0 ip 0000000000400535
sp 00007ffce4a80610 error 5 in reader
Let’s see whether this works...

- Adapted code

```c
*(volatile char*)0;
array[84 * 4096] = 0; // unreachable
```
Let’s see whether this works...

- Adapted code

```c
*(volatile char*)0;
array[84 * 4096] = 0; // unreachable
```

- Static code analyzer is not happy

```c
warning: Dereference of null pointer
  *(volatile char*)0;
```
Let’s see whether this works...

- Flush+Reload over all pages of the array

- “Unreachable” code line was actually executed
Let’s see whether this works...

- Flush+Reload over all pages of the array

  ![Chart](chart.png)

  - "Unreachable" code line was actually executed
  - Exception was only thrown afterwards

Daniel Gruss — Graz University of Technology
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
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;
```
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*)0xfffffffff81a000e0;
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
Leaking a picture like in CSI Cyber
Leaking Passwords from your Password Manager

Daniel Gruss — Graz University of Technology
DRAM organization
DRAM organization

channel 0

back of DIMM: rank 1

channel 1

front of DIMM: rank 0
DRAM organization

channel 0

back of DIMM: rank 1

channel 1

front of DIMM: rank 0

chip
# DRAM Organization

A DRAM chip contains multiple banks, each with a large number of rows. The figure below illustrates this organization:

- **Chip**: The entire DRAM module.
- **Bank 0**: One of the memory banks.
- **Row Buffer**: The buffer between the chip and the bank.
- **Rows**: Each bank contains a large number of rows, typically 32,768 rows in a 64k-cell bank.

<table>
<thead>
<tr>
<th>Bank 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>row 0</td>
</tr>
<tr>
<td>row 1</td>
</tr>
<tr>
<td>row 2</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>row 32767</td>
</tr>
<tr>
<td>row buffer</td>
</tr>
</tbody>
</table>

A 64k cell bank consists of 65,536 cells, each containing one capacitor and one transistor.
DRAM organization

chip

bank 0

- row 0
- row 1
- row 2
- ...
- row 32767

row buffer

64k cells
1 capacitor, 1 transistor each
- Cells leak $\rightarrow$ repetitive refresh necessary
- Maximum interval between refreshes to guarantee data integrity
- Cells leak faster upon proximate accesses $\rightarrow$ Rowhammer
• Cells leak $\rightarrow$ repetitive refresh necessary
• Maximum interval between refreshes to guarantee data integrity
• Cells leak faster upon proximate accesses $\rightarrow$ Rowhammer
- Cells leak $\rightarrow$ repetitive refresh necessary
- Maximum interval between refreshes to guarantee data integrity
- Cells leak faster upon proximate accesses $\rightarrow$ Rowhammer
• Cells leak $\rightarrow$ repetitive refresh necessary
• Maximum interval between refreshes to guarantee data integrity
• Cells leak faster upon proximate accesses $\rightarrow$ Rowhammer
Cells leak → repetitive refresh necessary

Max. interval between refreshes to guarantee data integrity

Cells leak faster upon proximate accesses → 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

![Diagram of DRAM bank with binary values and activate signal]

Daniel Gruss — Graz University of Technology
#1 - Single-sided hammering

[Diagram of DRAM bank with binary data being activated]
#1 - Single-sided hammering

![Diagram of DRAM bank with activation inhibit]

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

[Diagram of a DRAM bank with binary values and an activate arrow pointing to the lower set of values]
#1 - Single-sided hammering

[Diagram of a DRAM bank with binary data and an activation process]

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

![DRAM bank diagram showing bit flips]

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

[Diagram of DRAM bank with binary representation of data and an 'activate' arrow]
#2 - Double-sided hammering

 activate

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

#2 - Double-sided hammering

[Diagram of a DRAM bank with activate signal]
#2 - Double-sided hammering

![Diagram of DRAM bank with activation process]

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

![Diagram of DRAM bank with binary values and 'activate' indication]
#2 - Double-sided hammering

[Diagram of a DRAM bank with bit flips indicated]

- 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
```
#3 - One-location hammering

![DRAM bank diagram]

- Vertical lines:
  - 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  - 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  - 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  - 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  - 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  - 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  - 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  - 1 1 1 1 1 1 1 1 1 1 1 1 1 1

- Horizontal lines:
  - 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  - 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  - 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  - 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  - 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  - 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  - 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  - 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

- Highlighted line:
  - 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

- Bottom line:
  - 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

activate

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
#3 - One-location hammering

![Diagram of a DRAM bank with highlighted memory location]
#3 - One-location hammering

[Diagram of a DRAM bank with a highlighted row indicating activation]
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
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
Opcodes 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

0 1 1 1 0 1 0 0

PUSHQ

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

JE

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

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

0 1 1 1 0 1 0 0

JO

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

JE

0 1 1 1 0 1 0 0

JBE

0 1 1 1 0 1 1 0

Daniel Gruss — Graz University of Technology
Opcode Flipping - Conditional Jump

JE
01110100

JNE
01110101
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
We have ignored microarchitectural attacks for many many years:

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

Motor Vehicle Deaths in U.S. by Year

Seabelts
More Seabelts
Airbags
More Airbags
ABS
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?

Daniel Gruss — Graz University of Technology
• 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

- A 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?

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
What do we learn from it?

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