Rowhammer Attacks: A Walkthrough Guide

Daniel Gruss & Clémentine Maurice, Graz University of Technology

May 4, 2017 — RuhrSec 2017
Who are we

- **Daniel Gruss**
- PhD student @ Graz University Of Technology
- ✏️ @lavados
- 📧 daniel.gruss@iaik.tugraz.at
Who are we

- Clémentine Maurice
  - PhD in computer science, Postdoc @ Graz University Of Technology
  - @BloodyTangerine
  - clementine.maurice@iaik.tugraz.at
Goals of this talk

- you get a comprehensive overview of Rowhammer attacks
- you can run the tools on your machine
- you understand what’s happening and why

→ nothing here is black magic!
Outline

- Background
- How to flip bits?
- How to exploit them?
- How to mitigate them?
- Conclusion
1. Background
DRAM organization
DRAM organization

channel 0

channel 1
DRAM organization

back of DIMM: rank 1
front of DIMM: rank 0

canonical memory map

channel 0
channel 1
DRAM organization

- Back of DIMM: rank 1
- Front of DIMM: rank 0
- Channel 0
- Channel 1

Daniel Gruss & Clémentine Maurice, Graz University of Technology
May 4, 2017 — RuhrSec 2017
DRAM organization

- bits in cells in rows
- access: activate row, copy to row buffer

```
<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>
```
How reading from DRAM works

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

CPU wants to access row 1
How reading from DRAM works

CPU wants to access row 1
→ row 1 activated
How reading from DRAM works

CPU wants to access row 1
→ row 1 activated
→ row 1 copied to row buffer
How reading from DRAM works

CPU wants to access row 1
→ row 1 activated
→ row 1 copied to row buffer
How reading from DRAM works

DRAM bank

CPU wants to access row 2

row buffer
How reading from DRAM works

CPU wants to access row 2
→ row 2 activated
How reading from DRAM works

- **CPU wants to access row 2**
  - row 2 activated
  - row 2 copied to row buffer
How reading from DRAM works

CPU wants to access row 2
→ row 2 activated
→ row 2 copied to row buffer
How reading from DRAM works

CPU wants to access row 2
→ row 2 activated
→ row 2 copied to row buffer
→ slow (row conflict)
How reading from DRAM works

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

row buffer

CPU wants to access row 2—again
How reading from DRAM works

CPU wants to access row 2—again
→ row 2 already in row buffer
How reading from DRAM works

CPU wants to access row 2—again
→ row 2 already in row buffer
How reading from DRAM works

CPU wants to access row 2—again
→ row 2 already in row buffer
→ fast (row hit)
How reading from DRAM works

row buffer = cache
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
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
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

![Diagram of Rowhammer](image-url)
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
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

Diagram:

DRAM bank

activate

row buffer

copy
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
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
2. How to flip bits?
Requirements

Memory accesses must be

- **uncached**: reach DRAM
- **fast**: race against the next row refresh
- **targeted**: reach specific row
How do we get enough uncached accesses?
Impact of the CPU cache

- only non-cached accesses reach DRAM
Impact of the CPU cache

- only non-cached accesses reach DRAM
- either remove data from cache
Impact of the CPU cache

- only non-cached accesses reach DRAM
- either remove data from cache
- or don’t put it there in the first place
Impact of the CPU cache

- only non-cached accesses reach DRAM
- either remove data from cache
- or don’t put it there in the first place
  → next access will be served from DRAM
Access techniques

1. `clflush` instruction → original paper (Kim et al. 2014)
2. cache eviction (Gruss, Maurice, and Mangard 2016; Aweke et al. 2016)
3. non-temporal accesses (Qiao et al. 2016)
4. uncached memory (Veen et al. 2016)
#1 Hammering with `clflush`

cache set 1

cache set 2

DRAM bank
#1 Hammering with clflush

![Diagram showing cache sets and DRAM bank with clflush operations]
#1 Hammering with `clflush`

![Diagram showing hammering with clflush]

- Cache set 1
- Cache set 2

**DRAM bank**
#1 Hammering with clflush

```plaintext
cache set 1

cache set 2

DRAM bank
```
#1 Hammering with `clflush`

```plaintext

cache set 1

cache set 2

DRAM bank

reload

Daniel Gruss & Clémentine Maurice, Graz University of Technology
May 4, 2017 — RuhrSec 2017
```
#1 Hammering with clflush

cache set 1

DRAM bank

cache set 2

reload
#1 Hammering with clflush

cache set 1

cache set 2

DRAM bank

cflush

cflush
#1 Hammering with `clflush`

cache set 1

DRAM bank

reload

cache set 2

reload
#1 Hammering with clflush

![Diagram of cache sets and DRAM bank]

- **Cache set 1**
- **Cache set 2**
- **DRAM bank**
#1 Hammering with `clflush`
#1 Hammering with `clflush`

- cache set 1
- cache set 2

![DRAM bank diagram]
#1 Hammering with clflush

cache set 1

cache set 2

DRAM bank

reload

reload
#1 Hammering with \texttt{clflush}
#1 Hammering with \texttt{clflush}

cache set 1

cache set 2

DRAM bank

bit flip!

reload
How widespread is the issue?

**DDR3:**
- Kim et al.: 110/129 modules from 3 vendors, all but 3 since mid-2011
- Seaborn and Dullien: 15/29 laptops

**DDR4 believed to be safe:**
- we showed bit flips (Pessl et al. 2016)

Prevalence, by Kim et al. 2014
Flush, reload, flush, reload...

- the core of Rowhammer is essentially a Flush+Reload loop
- as much an attack on DRAM as on cache
#2 Hammering with cache eviction

- idea: avoid \texttt{clflush} to be independent of specific instructions
  - no \texttt{clflush} in JavaScript
#2 Hammering with cache eviction

- **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*!
#2 Hammering with cache eviction

- 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!
#2 Hammering with cache eviction

cache set 1

cache set 2

DRAM bank
#2 Hammering with cache eviction

cache set 1

load

load

cache set 2

DRAM bank
#2 Hammering with cache eviction

![Diagram showing cache eviction]

- Cache set 1
- Cache set 2
- DRAM bank

Daniel Gruss & Clémentine Maurice, Graz University of Technology
May 4, 2017 — RuhrSec 2017
#2 Hammering with cache eviction

- **Cache Set 1:**
  - Load

- **Cache Set 2:**
  - Load

- **DRAM Bank:**
  - Blocks

- **Legend:**
  - Gray: Evicted
  - Green: Loaded
#2 Hammering with cache eviction

cache set 1

load

cache set 2

load

DRAM bank
Hammering with cache eviction

- Cache set 1
- Cache set 2
- DRAM bank
#2 Hammering with cache eviction

`load`
#2 Hammering with cache eviction

![Diagram of cache sets and DRAM bank]

- cache set 1
- cache set 2
- DRAM bank

Daniel Gruss & Clémentine Maurice, Graz University of Technology
May 4, 2017 — RuhrSec 2017
#2 Hammering with cache eviction

DRAM bank

cache set 1

load

load

cache set 2
#2 Hammering with cache eviction

cache set 1

cache set 2

DRAM bank
#2 Hammering with cache eviction

repeat!

cache set 1

cache set 2

DRAM bank
#2 Hammering with cache eviction

![Diagram showing cache sets and cache eviction](image)

- Cache set 1
- Cache set 2

wait for it...
#2 Hammering with cache eviction

cache set 1

cache set 2

DRAM bank

bit flip!
Cache eviction strategies

Not as simple as that → replacement policy is not LRU
Cache eviction strategies

Not as simple as that → replacement policy is not LRU

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

Not as simple as that → replacement policy is not LRU

→ fast and effective on Haswell: eviction rate \( > 99.97\% \)
→ we evaluated 10,000+ strategies to find the best one
Hammering with cache eviction on Haswell

![Graph showing number of bit flips within 15 minutes against refresh interval in µs (BIOS configuration)]
#3 Hammering with non-temporal accesses

- non-temporal accesses: data accessed just once, not in the future
- NTA instructions \(\rightarrow\) bypass cache to minimize cache pollution
#3 Hammering with non-temporal accesses

- non-temporal accesses: data accessed just once, not in the future
- NTA instructions $\rightarrow$ bypass cache to minimize cache pollution
- NT stores to 1 address are combined at WC buffer
- only last write goes to DRAM $\rightarrow$ rate not sufficient
#3 Hammering with non-temporal accesses

- non-temporal accesses: data accessed just once, not in the future
- NTA instructions → bypass cache to minimize cache pollution
- NT stores to 1 address are combined at WC buffer
- only last write goes to DRAM → rate not sufficient
- following cached access to same address (Qiao et al. 2016)
#3 Hammering with non-temporal accesses

begin:
    movnti %eax, (X)
    movnti %eax, (Y)
    mov %eax, (X)
    mov %eax, (Y)
    jmp begin
#4 Hammering with uncached memory

Sometimes, everything fails,
#4 Hammering with uncached memory

Sometimes, everything fails, e.g., on mobile devices
#4 Hammering with uncached memory

Sometimes, everything fails, e.g., on mobile devices

- ARMv7 flush instruction is privileged
#4 Hammering with uncached memory

Sometimes, everything fails, e.g., on mobile devices

- ARMv7 flush instruction is privileged
- cache eviction seems to be too slow
#4 Hammering with uncached memory

Sometimes, everything fails, e.g., on mobile devices

- ARMv7 flush instruction is privileged
- cache eviction seems to be too slow
- ARMv8 non-temporal stores are still cached in practice
#4 Hammering with uncached memory

- **ION**: memory management since Android 4.0
- apps can use `/dev/ion` for **uncached**, physically contiguous memory
- no privilege and no permission needed (Veen et al. 2016)
How do we target accesses?
Physical addresses and DRAM

- fixed map: physical addresses $\rightarrow$ DRAM cells
- *undocumented* for Intel
- reverse-engineering for Sandy Bridge (Seaborne 2015)
- and by us for Sandy, Ivy, Haswell, Skylake, ... (Pessl et al. 2016)
- using the timing difference between row hits and row conflicts
How do I reverse my own DRAM?

🔗 https://github.com/IAIK/DRAMA

taskset 0x4 sudo ./measure -p 0.5 -s 16
# taskset core for stability
# sudo for pagemap access
# -p 0.5 allocate 50% of memory, the more the better
# -s I expect at least 16 sets (I have 32)
How do I flip bits?

https://github.com/IAIK/rowhammerjs

Copy functions from measure result

make ivy # or your microarchitecture
sudo ./rowhammer-ivy -d 2
# sudo for pagemap
# -d 2, for 2 DIMMs
sudo ./rowhammer-ivy -d 2 -f 0
# -f 0, only test offset 0 of every row
Demo

Demo!
3. How to exploit bit flips?
How to exploit random bit flips?

- They are not random $\rightarrow$ highly reproducible flip pattern!

1. choose a data structure that you can place at arbitrary memory locations
2. scan for “good” flips
3. place data structure there
4. trigger bit flip again
Strategy: Modify instructions

- idea from Seaborn and Dullien 2015
- x86 op codes are variable length
  - unsafe op codes (syscall) ∈ safe but long multi-byte op codes
  - only a problem with jumps to arbitrary addresses
- flip a bit in a validated NaCl instruction sequence
  - safe + validated jump → arbitrary jump
### Page Table Entries

<table>
<thead>
<tr>
<th>P</th>
<th>RW</th>
<th>US</th>
<th>WT</th>
<th>UC</th>
<th>R</th>
<th>D</th>
<th>S</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<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>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Page Table Entries

<table>
<thead>
<tr>
<th></th>
<th>P</th>
<th>RW</th>
<th>US</th>
<th>WT</th>
<th>UC</th>
<th>R</th>
<th>D</th>
<th>S</th>
<th>G</th>
<th>Ignored</th>
<th>Ignored</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<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>
<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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
## Page Table Entries

<table>
<thead>
<tr>
<th>P</th>
<th>RW</th>
<th>US</th>
<th>WT</th>
<th>UC</th>
<th>R</th>
<th>D</th>
<th>S</th>
<th>G</th>
<th>Ignored</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Physical Page Number**

<table>
<thead>
<tr>
<th>Ignored</th>
<th>X</th>
</tr>
</thead>
</table>

Ignored
### Page Table Entries

<table>
<thead>
<tr>
<th>P</th>
<th>RW</th>
<th>US</th>
<th>WT</th>
<th>UC</th>
<th>R</th>
<th>D</th>
<th>S</th>
<th>G</th>
<th>Ignored</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Physical Page Number**

<table>
<thead>
<tr>
<th>Ignored</th>
<th>X</th>
</tr>
</thead>
</table>

Each 4 KB page table consists of 512 such entries
Page Table Manipulation

Daniel Gruss & Clémentine Maurice, Graz University of Technology
May 4, 2017 — RuhrSec 2017
Page Table Manipulation
Page Table Manipulation

0x7000 – 0x7FFF
0x6000 – 0x6FFF
0x5000 – 0x5FFF
0x4000 – 0x4FFF
0x3000 – 0x3FFF
0x2000 – 0x2FFF
0x1000 – 0x1FFF
0x0 – 0xFFF

PTE 7
PTE 6
PTE 5
PTE 4
PTE 3
PTE 2
PTE 1
PTE 0

User Page
User Page
User Page
User Page
User Page
User Page
Kernel Page
Page Table
User Page
User Page
User Page
User Page
Page Table Manipulation

0x7000 – 0x7FFF
0x6000 – 0x6FFF
0x5000 – 0x5FFF
0x4000 – 0x4FFF
0x3000 – 0x3FFF
0x2000 – 0x2FFF
0x1000 – 0x1FFF
0x0 – 0xFFF

PTE 0
PTE 1
PTE 2
PTE 3
PTE 4
PTE 5
PTE 6
PTE 7

User Page
User Page
User Page
User Page
User Page
User Page
User Page
User Page

Kernel Page
Page Table

User Page
User Page
User Page
User Page
User Page
User Page
User Page
User Page

Page Table Manipulation

[Diagram showing page table entries and their corresponding pages]

Daniel Gruss & Clémantine Maurice, Graz University of Technology
May 4, 2017 — RuhrSec 2017
Page Table Manipulation

<table>
<thead>
<tr>
<th>Memory Address Range</th>
<th>PTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000 - 0x0FFF</td>
<td>PTE 0</td>
</tr>
<tr>
<td>0x1000 - 0x1FFF</td>
<td>PTE 1</td>
</tr>
<tr>
<td>0x2000 - 0x2FFF</td>
<td>PTE 2</td>
</tr>
<tr>
<td>0x3000 - 0x3FFF</td>
<td>PTE 3</td>
</tr>
<tr>
<td>0x4000 - 0x4FFF</td>
<td>PTE 4</td>
</tr>
<tr>
<td>0x5000 - 0x5FFF</td>
<td>PTE 5</td>
</tr>
<tr>
<td>0x6000 - 0x6FFF</td>
<td>PTE 6</td>
</tr>
<tr>
<td>0x7000 - 0x7FFF</td>
<td>PTE 7</td>
</tr>
</tbody>
</table>

- User Page
- Kernel Page
- Page Table
Page Table Manipulation

- 0x0 - 0xFFF (Page Table)
- 0x1000 - 0x1FFF (Kernel Page)
- 0x2000 - 0x2FFF
- 0x3000 - 0x3FFF
- 0x4000 - 0x4FFF
- 0x5000 - 0x5FFF
- 0x6000 - 0x6FFF
- 0x7000 - 0x7FFF
Search for page with flip

Hammering memory locations in different rows
Search for page with flip

Hammering memory locations in different rows
Search for page with flip

Hammering memory locations in different rows
Search for page with flip

Hammering memory locations in different rows
Search for page with flip

Hammering memory locations in different rows
Search for page with flip

Hammering memory locations in different rows
Release page with flip
Fill all remaining memory with page tables
Fill all remaining memory with page tables
Page Table Manipulation

![Page Table Diagram]

- Pages range from $0x0000$ to $0xFFFF$
- Pages are divided into User Page and Kernel Page
- Page Table entries are labeled PTE 0 to PTE 7

Daniel Gruss & Clémentine Maurice, Graz University of Technology
May 4, 2017 — RuhrSec 2017
Page Table Manipulation
Page Table Manipulation

<table>
<thead>
<tr>
<th>Address Range</th>
<th>PTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x7000 – 0x7FFF</td>
<td>PTE 7</td>
</tr>
<tr>
<td>0x6000 – 0x6FFF</td>
<td>PTE 6</td>
</tr>
<tr>
<td>0x5000 – 0x5FFF</td>
<td>PTE 5</td>
</tr>
<tr>
<td>0x4000 – 0x4FFF</td>
<td>PTE 4</td>
</tr>
<tr>
<td>0x3000 – 0x3FFF</td>
<td>PTE 3</td>
</tr>
<tr>
<td>0x2000 – 0x2FFF</td>
<td>PTE 2</td>
</tr>
<tr>
<td>0x1000 – 0x1FFF</td>
<td>PTE 1</td>
</tr>
<tr>
<td>0x0 – 0xFFF</td>
<td>PTE 0</td>
</tr>
</tbody>
</table>

- Page Table
- User Page
- Kernel Page
Strategy: Flipping Page Table PPN bits

1. scan for flips
2. exhaust or massage memory to place a page table at target location
3. gain access to your own page table → kernel privileges
Flipping Page Table PPN bits

- idea from Seaborn and Dullien 2015
- same idea applied in several other works:
  - Rowhammer.js (Gruss, Maurice, and Mangard 2016)
  - One bit flips, one cloud flops (Xiao et al. 2016)
  - Drammer (Veen et al. 2016)
Post-Rowhammer Exploitation

- scan entire physical memory (very fast) and:
  - modify binary pages executed in root privileges (Xiao et al. 2016)
  - modify credential structs (Veen et al. 2016)
  - read keys (Xiao et al. 2016)
  - corrupt RSA signatures (Bhattacharya et al. 2016)
  - modify certificates
  - configurations
  - etc.

- pages are pretty unique: 32768 bits per page
Bit Flips + Page Deduplication
Bit Flips + Page Deduplication

Page with bit flip is filled with target content
Bit Flips + Page Deduplication

OS or hypervisor searches for duplicate pages
Bit Flips + Page Deduplication

OS or hypervisor searches for duplicate pages
Bit Flips + Page Deduplication

OS or hypervisor searches for duplicate pages
Bit Flips + Page Deduplication

OS or hypervisor searches for duplicate pages
Bit Flips + Page Deduplication

OS or hypervisor searches for duplicate pages
Bit Flips + Page Deduplication

OS or hypervisor searches for duplicate pages
Bit Flips + Page Deduplication

<table>
<thead>
<tr>
<th>Row 0</th>
<th>Row 23</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image-url" alt="Grid Image" /></td>
<td></td>
</tr>
</tbody>
</table>

OS or hypervisor searches for duplicate pages
Bit Flips + Page Deduplication

OS or hypervisor searches for duplicate pages
Bit Flips + Page Deduplication

| Row 0 | Row 1 | Row 2 | Row 3 | Row 4 | Row 5 | Row 6 | Row 7 | Row 8 | Row 9 | Row 10 | Row 11 | Row 12 | Row 13 | Row 14 | Row 15 | Row 16 | Row 17 | Row 18 | Row 19 | Row 20 | Row 21 | Row 22 | Row 23 |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|

Hammer again + flip again
Bit Flips + Page Deduplication

<table>
<thead>
<tr>
<th>Row 0</th>
<th>Row 23</th>
</tr>
</thead>
</table>

Daniel Gruss & Clémentine Maurice, Graz University of Technology
May 4, 2017 — RuhrSec 2017
Strategy: Flipping in Deduplicated Pages

1. scan for flips
2. place content for deduplication so that flip can be exploited
3. perform the bit change through Rowhammer
Flipping in Deduplicated Pages

- idea from Bosman et al. 2016
  - change data type (double → pointer)
  - change pointer to good object → counterfeit object
- and from Razavi et al. 2016
  - corrupt authorized SSH keys
  - corrupt Debian update URLs + RSA public key file
4. How to mitigate Rowhammer?
Mitigations

Different mitigations have been proposed:

- detection vs prevention
- software vs hardware
- short-term vs long-term
Quick fixes

- no `clflush` instruction
Quick fixes

- no `clflush` instruction → Rowhammer.js
Quick fixes

- no `clflush` instruction → Rowhammer.js
- increase the refresh rate
Quick fixes

- no `clflush` instruction → Rowhammer.js
- increase the refresh rate
  → would need to be increased by $7 \times$ to eliminate all bit flips

Errors depending on refresh interval (Kim et al. 2014)
Quick fixes

- no `clflush` instruction → `Rowhammer.js`
- increase the refresh rate
  → would need to be increased by $7 \times$ to eliminate all bit flips
  → implementation: increased by $2 \times$ by BIOS vendors

Errors depending on refresh interval (Kim et al. 2014)
What about ECC?

- ECC protection: server can handle or correct single bit errors
What about ECC?

- ECC protection: server can handle or correct single bit errors
- no standard for event reporting
What about ECC?

- ECC protection: server can handle or correct single bit errors
- **no standard** for event reporting
- in practice (Lanteigne 2016)
  - common: server counts ECC errors and report only if they reach a threshold (e.g., > 100 bit flips / hour)
What about ECC?

- ECC protection: server can handle or correct single bit errors
- no standard for event reporting
- in practice (Lanteigne 2016)
  - common: server counts ECC errors and report only if they reach a threshold (e.g., > 100 bit flips / hour)
  - some server vendors never report errors to the OS
What about ECC?

- ECC protection: server can handle or correct single bit errors
- no standard for event reporting
- in practice (Lanteigne 2016)
  - common: server counts ECC errors and report only if they reach a threshold (e.g., > 100 bit flips / hour)
  - some server vendors never report errors to the OS
  - one server did not even halt when bit flips were non-correctable
Detecting Rowhammer attacks

- Rowhammer: lots of cache misses that can be monitored with hardware performance counters (Herath et al. 2015; Gruss, Maurice, Wagner, et al. 2016; Chiappetta et al. 2015; Payer 2016)
Preventing Rowhammer attacks in hardware (1/3)

Original ideas from Kim et al. 2014

- making better DRAM chips that are not vulnerable,
- using error correcting codes (ECC)
- increasing the refresh rate
- remapping/retiring faulty cells after manufacturing
- identifying hammered rows at runtime and refreshing neighbors
Preventing Rowhammer attacks in hardware (1/3)

Original ideas from Kim et al. 2014

- making better DRAM chips that are not vulnerable,
- using error correcting codes (ECC)
- increasing the refresh rate
- remapping/retiring faulty cells after manufacturing
- identifying hammered rows at runtime and refreshing neighbors

→ expensive, performance overhead, or increased power consumption
Preventing Rowhammer attacks in hardware (2/3)

PARA - Probabilistic Adjacent Row Activation (Kim et al. 2014)

- one row closed → one adjacent row opened with low probability $p$
Preventing Rowhammer attacks in hardware (2/3)

PARA - Probabilistic Adjacent Row Activation (Kim et al. 2014)

- one row closed → one adjacent row opened with low probability $p$
- Rowhammer: one row opened and closed a high number of times $N_{th}$
Preventing Rowhammer attacks in hardware (2/3)

PARA - Probabilistic Adjacent Row Activation (Kim et al. 2014)

- one row closed $\rightarrow$ one adjacent row opened with low probability $p$
- Rowhammer: one row opened and closed a high number of times $N_{th}$
- statistically, neighbor rows are refreshed $\rightarrow$ no bit flip
Preventing Rowhammer attacks in hardware (2/3)

PARA - Probabilistic Adjacent Row Activation (Kim et al. 2014)

- one row closed $\rightarrow$ one adjacent row opened with low probability $p$
- Rowhammer: one row opened and closed a high number of times $N_{th}$
- statistically, neighbor rows are refreshed $\rightarrow$ no bit flip
- implementation at the memory controller level
Preventing Rowhammer attacks in hardware (2/3)

PARA - Probabilistic Adjacent Row Activation (Kim et al. 2014)

- one row closed $\rightarrow$ one adjacent row opened with low probability $p$
- Rowhammer: one row opened and closed a high number of times $N_{th}$
- statistically, neighbor rows are refreshed $\rightarrow$ no bit flip
- implementation at the memory controller level
- advantage: stateless $\rightarrow$ not expensive

Daniel Gruss & Clémentine Maurice, Graz University of Technology
May 4, 2017 — RuhrSec 2017
Preventing Rowhammer attacks in hardware (2/3)

PARA - Probabilistic Adjacent Row Activation (Kim et al. 2014)

- one row closed $\rightarrow$ one adjacent row opened with low probability $p$
- Rowhammer: one row opened and closed a high number of times $N_{th}$
- statistically, neighbor rows are refreshed $\rightarrow$ no bit flip
- implementation at the memory controller level
- advantage: stateless $\rightarrow$ not expensive
- for $p = 0.001$ and $N_{th} = 100K$, experiencing one error in one year has a probability $9.4 \times 10^{-14}$
Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold
Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold
Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold
Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold
Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold
Target Row Refresh (TRR)

- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold
Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold
Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold
Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold
Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold
Target Row Refresh (TRR)

- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold
Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold
Preventing Rowhammer attacks in software

“nohammer” kernel module Corbet 2016

- refresh rate of 8 ms would prevent Rowhammer on most systems
- use PMC to measure cache misses per 64 ms interval
- limit cache miss rate to 1/8 of maximum
Preventing Rowhammer attacks in software

“nohammer” kernel module Corbet 2016

- refresh rate of 8 ms would prevent Rowhammer on most systems
- use PMC to measure cache misses per 64 ms interval
- limit cache miss rate to 1/8 of maximum
Preventing Rowhammer attacks in software

“nohammer” kernel module Corbet 2016

- refresh rate of 8 ms would prevent Rowhammer on most systems
- use PMC to measure cache misses per 64 ms interval
- limit cache miss rate to 1/8 of maximum
Preventing Rowhammer attacks in software

“nohammer” kernel module Corbet 2016

- refresh rate of 8 ms would prevent Rowhammer on most systems
- use PMC to measure cache misses per 64 ms interval
- limit cache miss rate to 1/8 of maximum

Wait for refresh
Preventing Rowhammer attacks in software

“nohammer” kernel module Corbet 2016

- refresh rate of 8 ms would prevent Rowhammer on most systems
- use PMC to measure cache misses per 64 ms interval
- limit cache miss rate to 1/8 of maximum

Wait for refresh

Wait for refresh
Preventing Rowhammer attacks in software

“nohammer” kernel module Corbet 2016

- refresh rate of 8 ms would prevent Rowhammer on most systems
- use PMC to measure cache misses per 64 ms interval
- limit cache miss rate to 1/8 of maximum

Wait for refresh
Preventing Rowhammer attacks in software

“nohammer” kernel module Corbet 2016

- refresh rate of 8 ms would prevent Rowhammer on most systems
- use PMC to measure cache misses per 64 ms interval
- limit cache miss rate to 1/8 of maximum
Preventing Rowhammer attacks in software

“nohammer” kernel module Corbet 2016

- refresh rate of 8 ms would prevent Rowhammer on most systems
- use PMC to measure cache misses per 64 ms interval
- limit cache miss rate to 1/8 of maximum
Preventing Rowhammer attacks in software

“nohammer” kernel module Corbet 2016

- refresh rate of 8 ms would prevent Rowhammer on most systems
- use PMC to measure cache misses per 64 ms interval
- limit cache miss rate to 1/8 of maximum
Preventing Rowhammer attacks in software

MASCAT - Stopping Microarchitectural Attacks Before Execution (Irazoqui et al. 2016)

- static analysis of the binary
- detect suspicious instruction sequences (clflush, rdtsc, fences, ...)
- open problem: false positives
Preventing Rowhammer attacks in software

ANVIL (Aweke et al. 2016)

- uses performance counters to detect rowhammer
- activate rows neighbor rows to prevent flips
- similar as PARA, but in software
Preventing Rowhammer attacks in software

ANVIL (Aweke et al. 2016)

- uses performance counters to detect rowhammer
- activate rows neighbor rows to prevent flips
- similar as PARA, but in software
Preventing Rowhammer attacks in software

- B-CATT: disable vulnerable physical memory (Brasser et al. 2016)
- G-CATT: isolate security domains in physical memory based on potential vulnerability (Brasser et al. 2016)
Preventing Rowhammer attacks in software

- **B-CATT**: disable vulnerable physical memory (Brasser et al. 2016)
- **G-CATT**: isolate security domains in physical memory based on potential vulnerability (Brasser et al. 2016)
5. Conclusion
Conclusion

- Rowhammer attacks are easy to mount
- works on most systems (if you know the DRAM mapping)
- most countermeasures are too expensive or ineffective
I want to try!

- [https://github.com/IAIK/DRAMA](https://github.com/IAIK/DRAMA)
  Reverse-engineering tool for DRAM addressing

- [https://github.com/IAIK/rowhammerjs](https://github.com/IAIK/rowhammerjs)
  Adaptation of double-sided hammering + hammering in JavaScript

- [https://github.com/IAIK/armageddon](https://github.com/IAIK/armageddon)
  libflush provides performant eviction strategies

- [https://github.com/vusec/drammer](https://github.com/vusec/drammer)
  Hammering with ION on ARM
Thank you!

Contact

@lavados
@BloodyTangerine
Rowhammer Attacks: A Walkthrough Guide

Daniel Gruss & Clémentine Maurice, Graz University of Technology

May 4, 2017 — RuhrSec 2017
References I


Brasser, Ferdinand, Lucas Davi, David Gens, Christopher Liebchen, and Ahmad-Reza Sadeghi (2016). “CAn’t Touch This: Practical and Generic Software-only Defenses Against Rowhammer Attacks”. In: arXiv:1611.08396.


References V


