Micro-Architectural Fault Attacks

Clémentine Maurice, CNRS

May 5, 2019—FICHSA, Ben-Gurion University of the Negev, Beer Sheva, Israel

Fault attacks

- side channels: non-legitimate “read primitive”

Fault attacks

- side channels: non-legitimate “read primitive”
- can we have a “write primitive”?

Fault attacks

- side channels: non-legitimate “read primitive”
- can we have a “write primitive”?
- yes! fault attacks!

Fault attacks

- side channels: non-legitimate “read primitive”
- can we have a “write primitive”?
- yes! fault attacks!
- why is that a problem?

Fault attacks

- side channels: non-legitimate “read primitive”
- can we have a “write primitive”?
- yes! fault attacks!
- why is that a problem? hardware can bypass software security mechanisms
  $\rightarrow$ software cannot trust hardware anymore

Software-based fault attacks?

- until 2014, fault attacks required physical access: changes in temperature, clock, voltage, electric/magnetic fields, ...

Yoongu Kim et al. “Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors”. In: *ISCA'14*. 2014.

Software-based fault attacks?

- until 2014, fault attacks required physical access: changes in temperature, clock, voltage, electric/magnetic fields, ...
- core idea of fault attacks: pushing hardware beyond nominal operating conditions

Yoongu Kim et al. "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors". In: *ISCA'14*. 2014.

Software-based fault attacks?

- until 2014, fault attacks required physical access: changes in temperature, clock, voltage, electric/magnetic fields, ...
- core idea of fault attacks: pushing hardware beyond nominal operating conditions
- software can do that too!

Yoongu Kim et al. "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors". In: *ISCA'14*. 2014.

Software-based fault attacks?

- until 2014, fault attacks required physical access: changes in temperature, clock, voltage, electric/magnetic fields, ...
- core idea of fault attacks: pushing hardware beyond nominal operating conditions
- software can do that too!
- 2014: Rowhammer (Kim et al.)

Yoongu Kim et al. "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors". In: *ISCA'14*. 2014.

Software-based fault attacks?

- until 2014, fault attacks required physical access: changes in temperature, clock, voltage, electric/magnetic fields, ...
- core idea of fault attacks: pushing hardware beyond nominal operating conditions
- software can do that too!
- 2014: Rowhammer (Kim et al.)
- first reaction: "it's a reliability issue, not a security issue"

Yoongu Kim et al. "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors". In: *ISCA'14*. 2014.

Software-based fault attacks?

- until 2014, fault attacks required physical access: changes in temperature, clock, voltage, electric/magnetic fields, ...
- core idea of fault attacks: pushing hardware beyond nominal operating conditions
- software can do that too!
- 2014: Rowhammer (Kim et al.)
- first reaction: "it's a reliability issue, not a security issue"
- 2015: Google Project Zero showed a sandbox escape and a privilege escalation attack using Rowhammer

---

Yoongu Kim et al. "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors". In: *ISCA'14*. 2014.

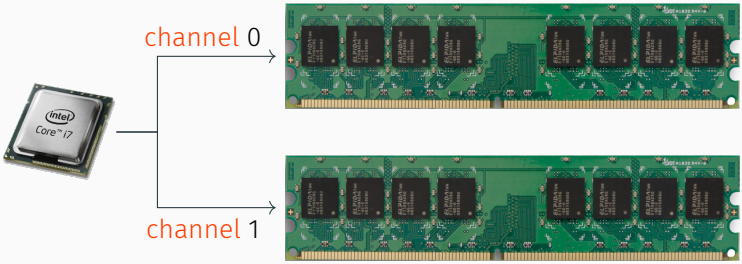
Outline

- Background on DRAM and Rowhammer
- How do we get bip flips?
- How do we target memory accesses?
- Can we exploit these bit flips?
- Countermeasures

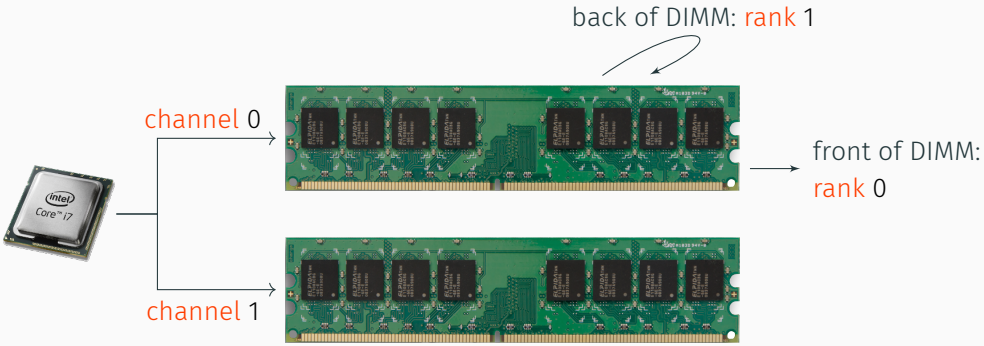
Background on DRAM and Rowhammer

DRAM organization

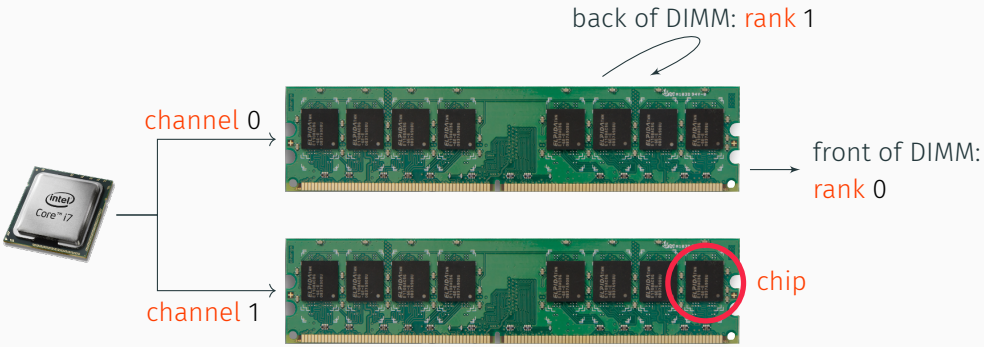
DRAM organization



DRAM organization



DRAM organization



DRAM organization

chip

bank 0

row 0
row 1
row 2
...
row 32767

row buffer

DRAM organization

chip

bank 0

row 0
row 1
row 2
...
row 32767

row buffer

64k cells
1 capacitor,
1 transitor each

DRAM row buffer

- DRAM internally is only capable of reading entire rows

DRAM row buffer

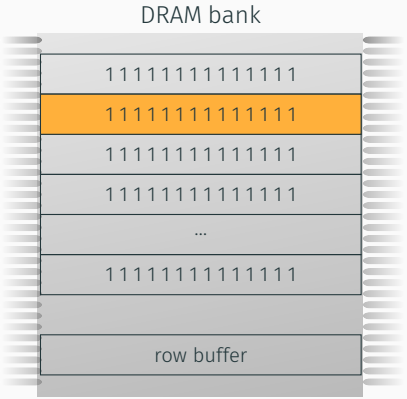
- DRAM internally is only capable of reading entire rows
- capacitors in cells discharge when you “read the bits”
- buffer the bits when reading them from the cells
- write the bits back to the cells when you’re done

DRAM row buffer

- DRAM internally is only capable of reading entire rows
- capacitors in cells discharge when you “read the bits”
- buffer the bits when reading them from the cells
- write the bits back to the cells when you’re done

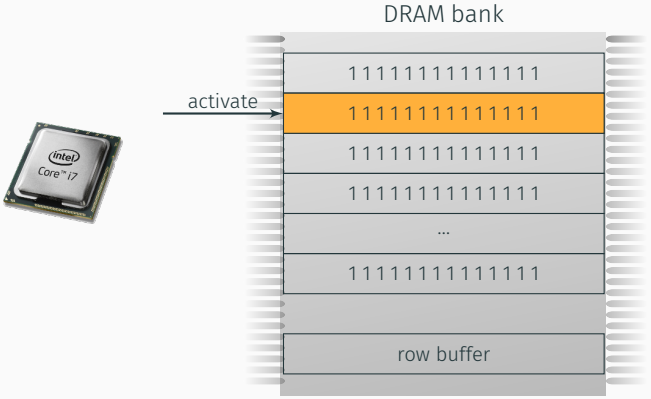
→ row buffer

How reading from DRAM works



CPU wants to access row 1

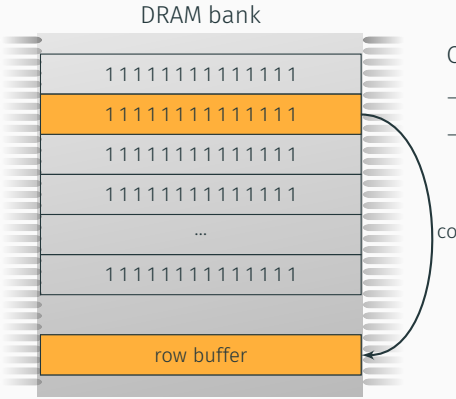
How reading from DRAM works



CPU wants to access row 1

→ row 1 activated

How reading from DRAM works

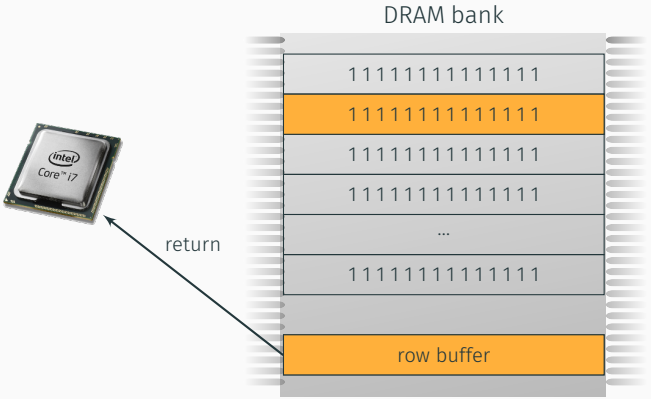


CPU wants to access row 1

→ row 1 activated

→ row 1 copied to row buffer

How reading from DRAM works

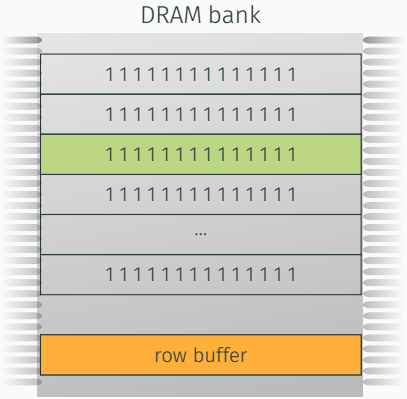


CPU wants to access row 1

→ row 1 activated

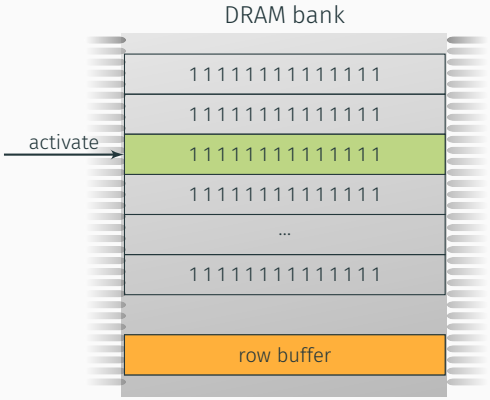
→ row 1 copied to row buffer

How reading from DRAM works



CPU wants to access row 2

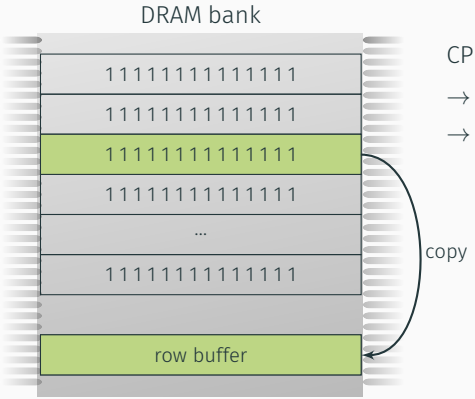
How reading from DRAM works



CPU wants to access row 2

$\rightarrow$ row 2 activated

How reading from DRAM works

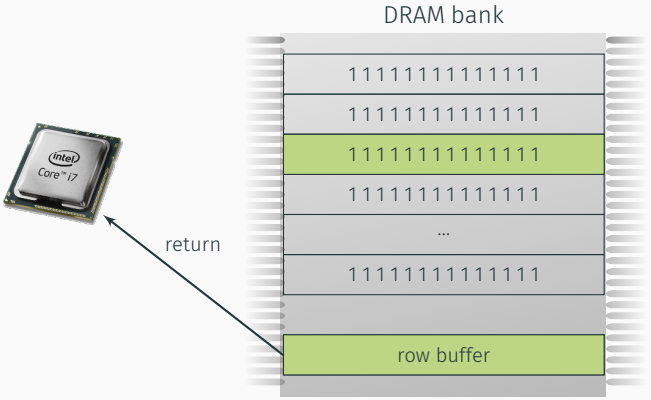


CPU wants to access row 2

→ row 2 activated

→ row 2 copied to row buffer

How reading from DRAM works

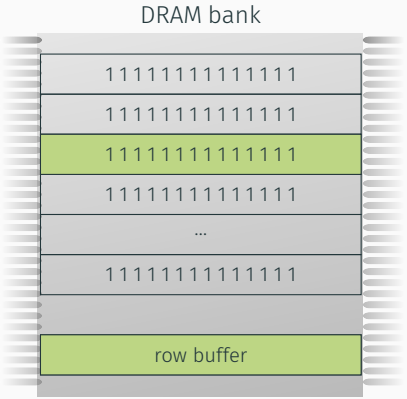


CPU wants to access row 2

→ row 2 activated

→ row 2 copied to row buffer

How reading from DRAM works



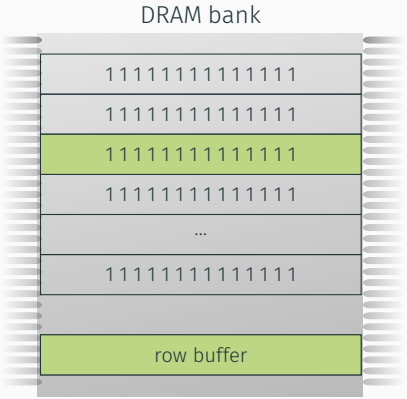
CPU wants to access row 2

→ row 2 activated

→ row 2 copied to row buffer

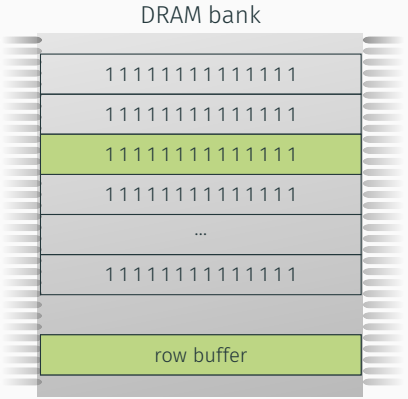
→ slow (row conflict)

How reading from DRAM works



CPU wants to access row 2—again

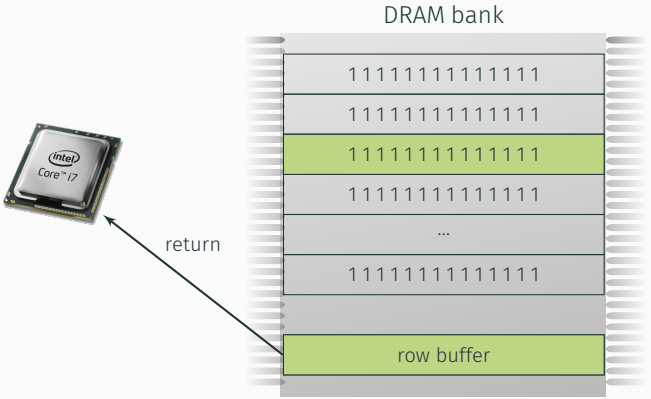
How reading from DRAM works



CPU wants to access row 2—again

→ row 2 already in row buffer

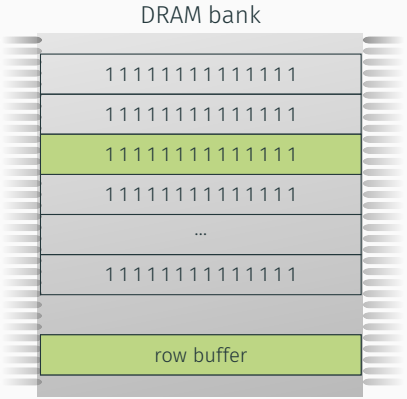
How reading from DRAM works



CPU wants to access row 2—again

→ row 2 already in row buffer

How reading from DRAM works

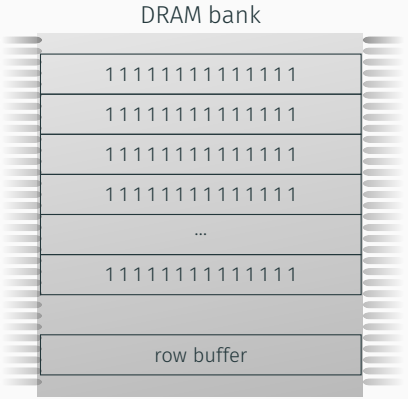


CPU wants to access row 2—again

→ 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 → repetitive refresh necessary
- refresh ≈ reading (destructive) + writing same data again
- maximum interval between refreshes to guarantee data integrity

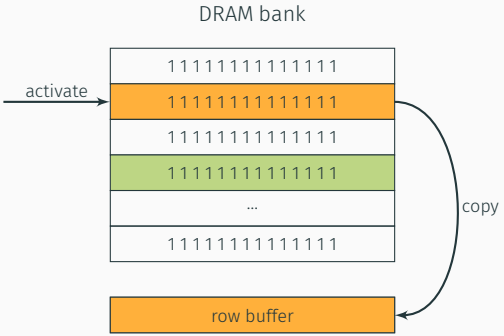
- cells leak faster upon proximate accesses → fault attack

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

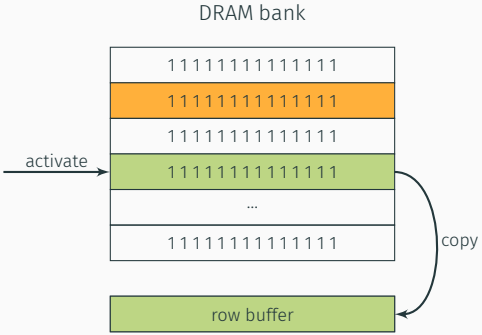
Rowhammer

"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



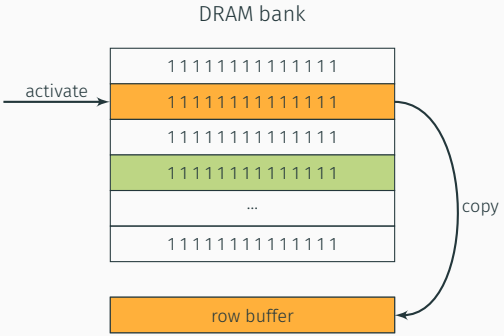
Rowhammer

"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



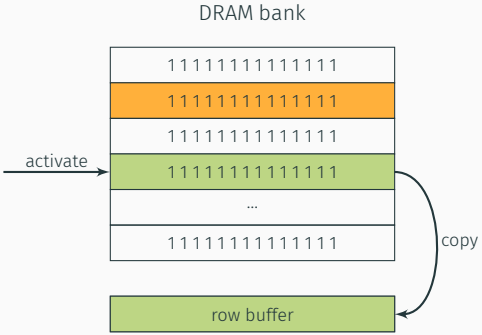
Rowhammer

"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



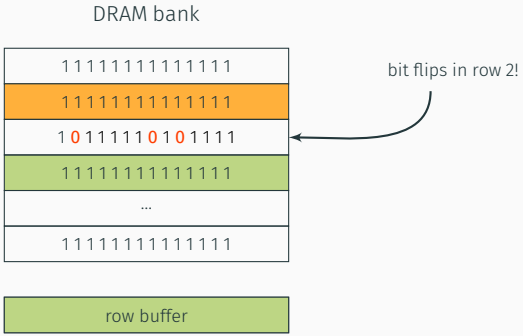
Rowhammer

"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



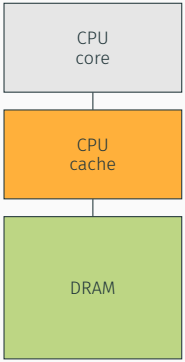
Requirements

Memory accesses must be

- uncached: reach DRAM
- fast: race against the next row refresh
- targeted: reach specific row

Issue #1: How do we get bit flips?

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

How to reach DRAM?

1. `clflush` instruction → original paper (Kim et al.)
2. cache eviction (Gruss et al., Aweke et al.)
3. non-temporal accesses (Qiao et al.)
4. uncached memory (van der Veen et al.)
5. remotely (Lipp et al., Tatar et al.)

Yoongu Kim et al. “Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors”. In: *ISCA’14*. 2014.

Daniel Gruss et al. “Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript”. In: *DIMVA’16*. 2016.

Zelalem Birhanu Aweke et al. “ANVIL: Software-based protection against next-generation rowhammer attacks”. In: *ACM SIGPLAN Notices* 51.4 (2016), pp. 743–755.

Rui Qiao et al. “A new approach for rowhammer attacks”. In: *HOST 2016*. 2016.

Victor van der Veen et al. “Drammer: Deterministic Rowhammer Attacks on Mobile Platforms”. In: *CCS’16*. 2016.

Moritz Lipp et al. “Nethammer: Inducing Rowhammer Faults through Network Requests”. In: *arXiv:1805.04956* (2018).

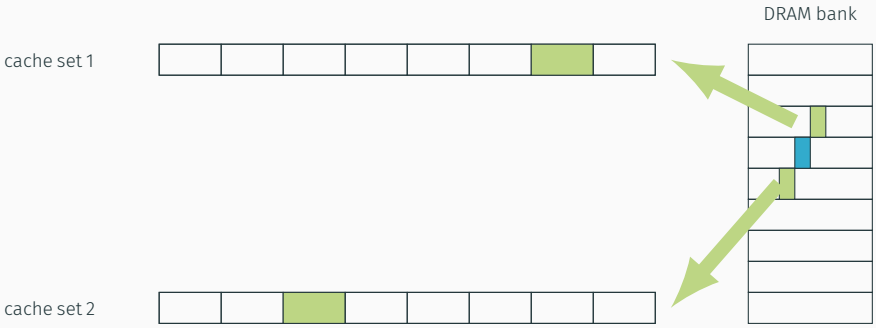
Andrei Tatar et al. “Throwhammer: Rowhammer Attacks over the Network and Defenses”. In: *USENIX ATC 2018*. 2018.

#1 Hammering with `clflush`

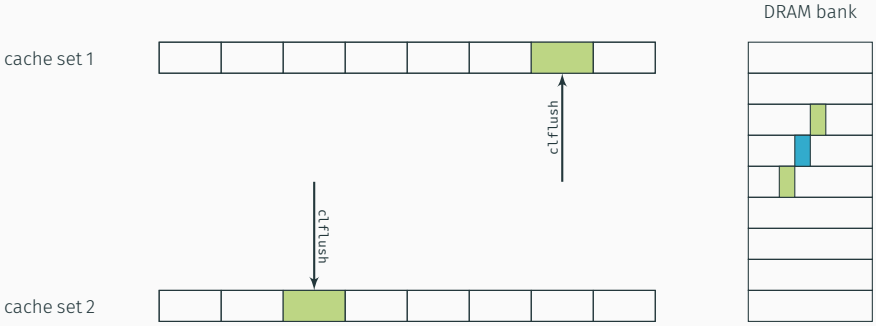
```
begin:
  mov (X), %eax   // read from address X
  mov (Y), %ebc   // read from address Y
  clflush (X)     // flush cache for address X
  clflush (Y)     // flush cache for address Y
  jmp begin
```

Yoongu Kim et al. “Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors”. In: *ISCA’14*. 2014.

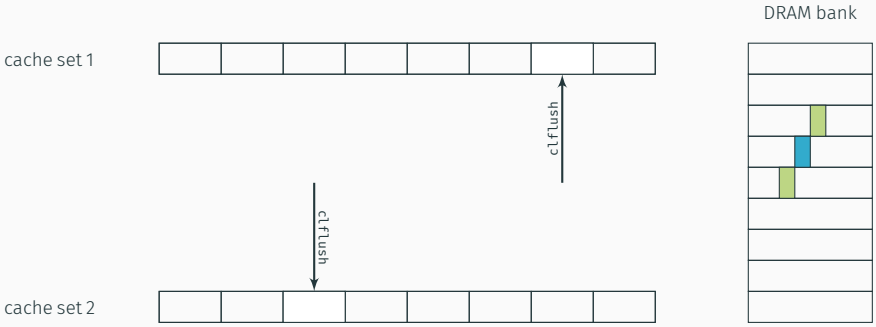
#1 Hammering with `clflush`



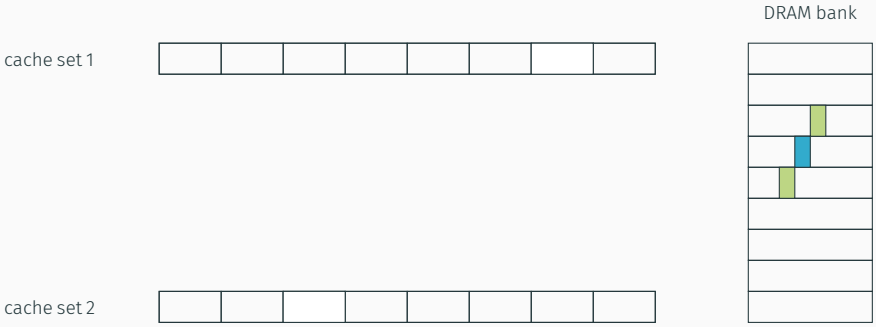
#1 Hammering with `clflush`



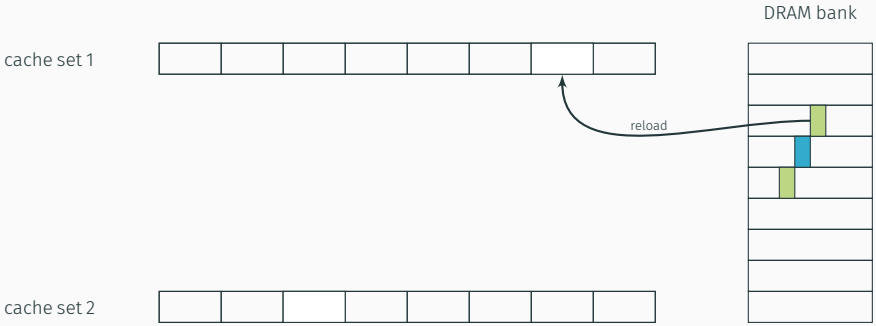
#1 Hammering with `clflush`



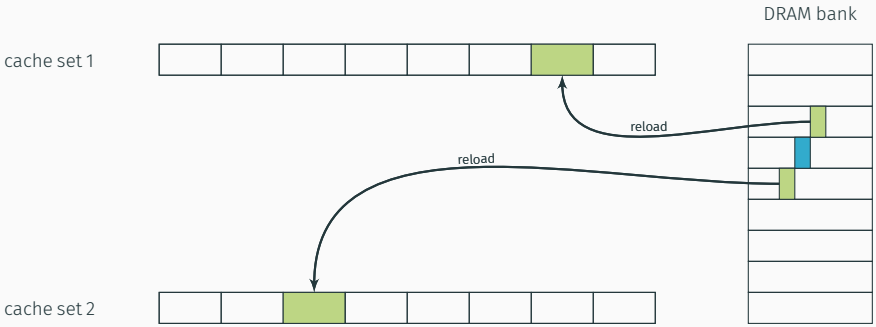
#1 Hammering with `clflush`



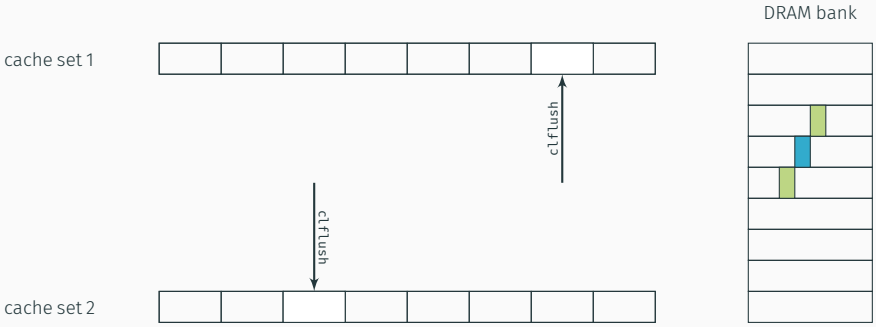
#1 Hammering with `clflush`



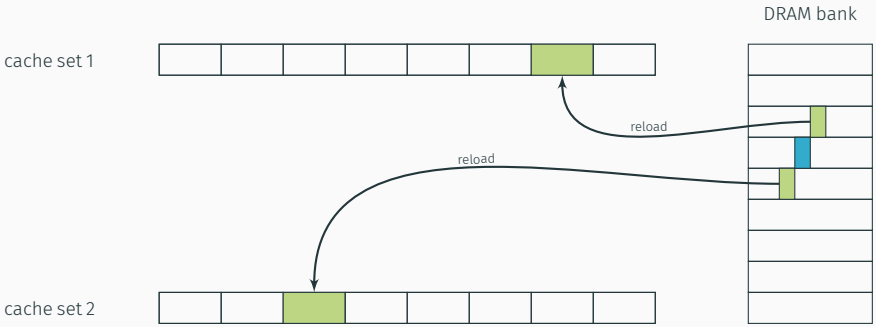
#1 Hammering with `clflush`



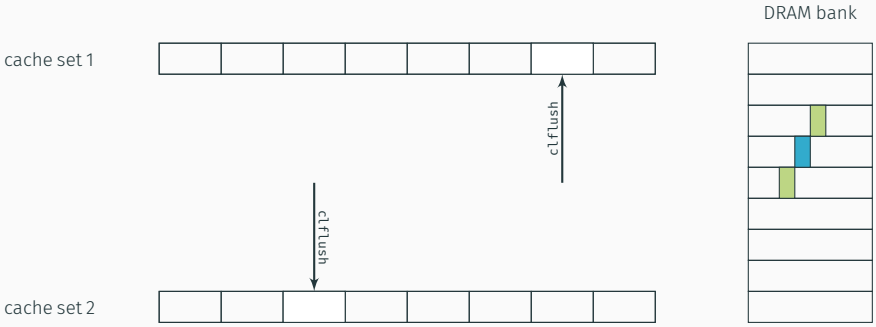
#1 Hammering with `clflush`



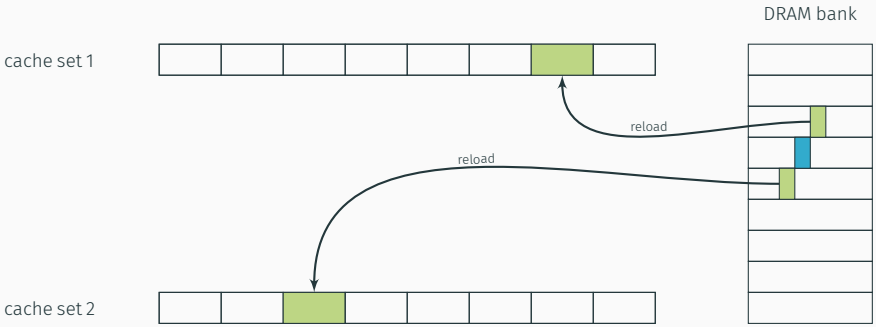
#1 Hammering with `clflush`



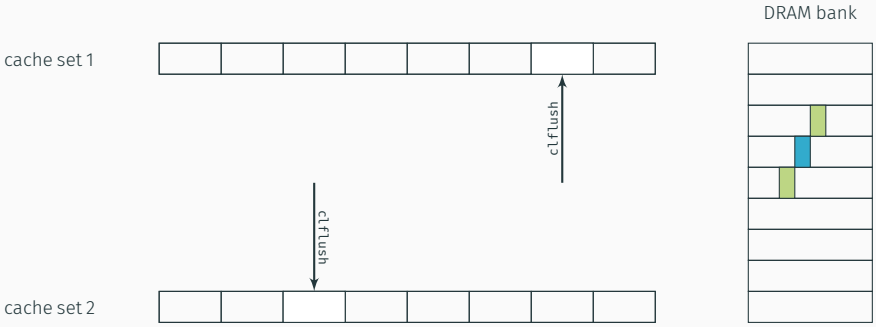
#1 Hammering with `clflush`



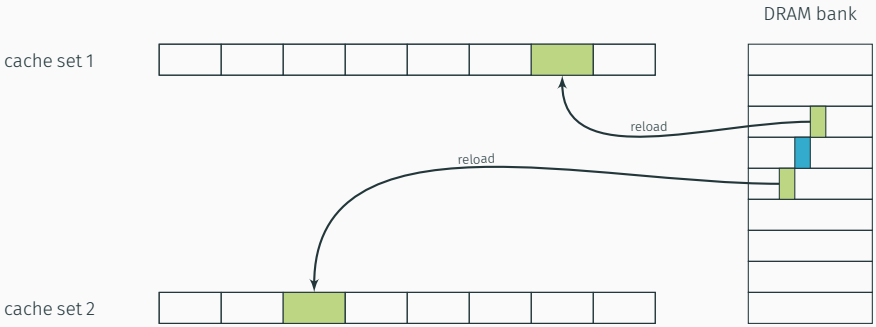
#1 Hammering with `clflush`



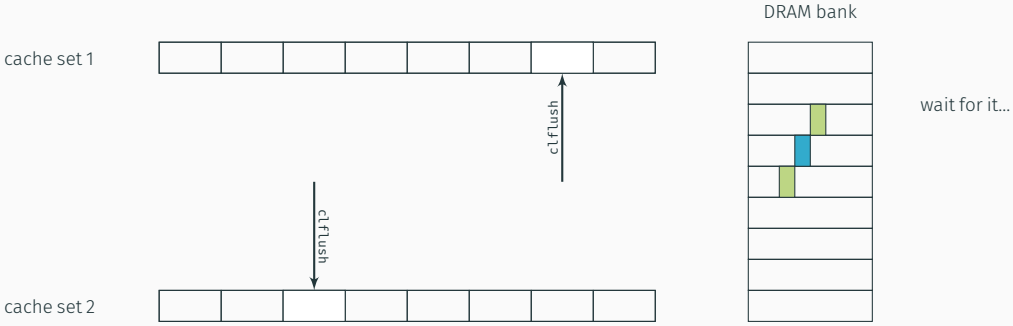
#1 Hammering with `clflush`



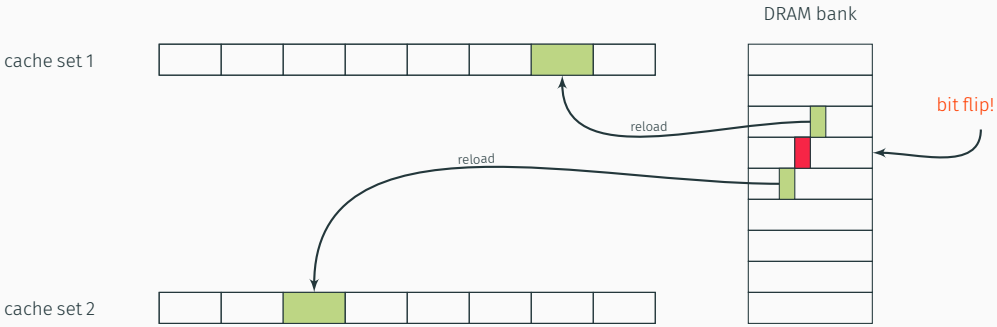
#1 Hammering with `clflush`



#1 Hammering with `clflush`



#1 Hammering with `clflush`



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 `clflush` to be independent of specific instructions
  → no `clflush` in JavaScript

Daniel Gruss et al. “Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript”. In: *DIMVA'16*. 2016.

#2 Hammering with cache eviction

- idea: avoid `clflush` to be independent of specific instructions
  → no `clflush` in JavaScript
- what can we do?

Daniel Gruss et al. “Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript”. In: *DIMVA'16*. 2016.

#2 Hammering with cache eviction

- idea: avoid `clflush` to be independent of specific instructions
  - → no `clflush` in JavaScript
- what can we do?
- our approach: use regular memory accesses for eviction
  - → techniques from cache attacks
  - → Rowhammer using Prime+Probe

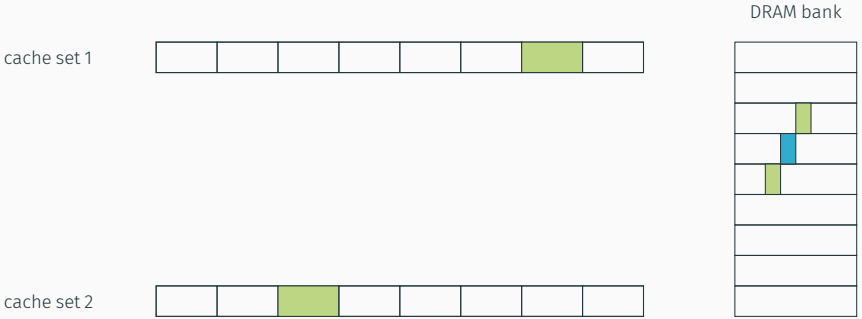
Daniel Gruss et al. "Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript". In: *DIMVA'16*. 2016.

#2 Hammering with cache eviction

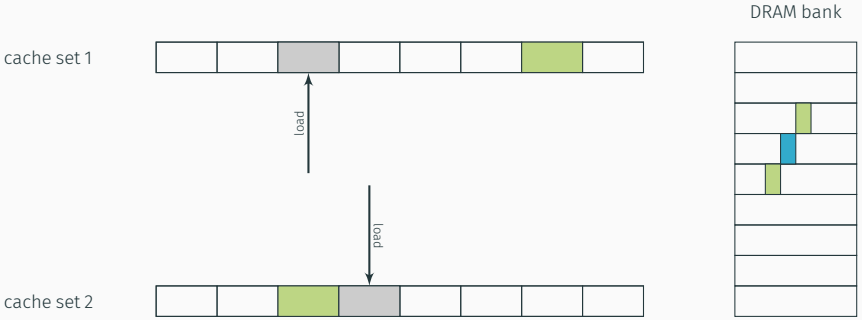
- idea: avoid `clflush` to be independent of specific instructions
  - → no `clflush` in JavaScript
- what can we do?
- our approach: use regular memory accesses for eviction
  - → techniques from cache attacks
  - → Rowhammer using Prime+Probe
- beware of the replacement policy!

Daniel Gruss et al. "Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript". In: *DIMVA'16*. 2016.

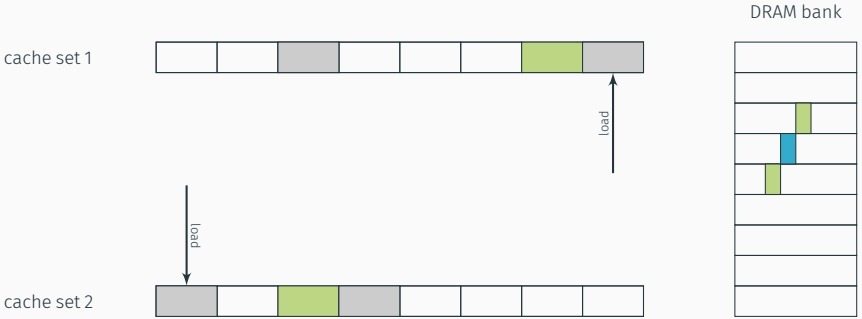
#2 Hammering with cache eviction



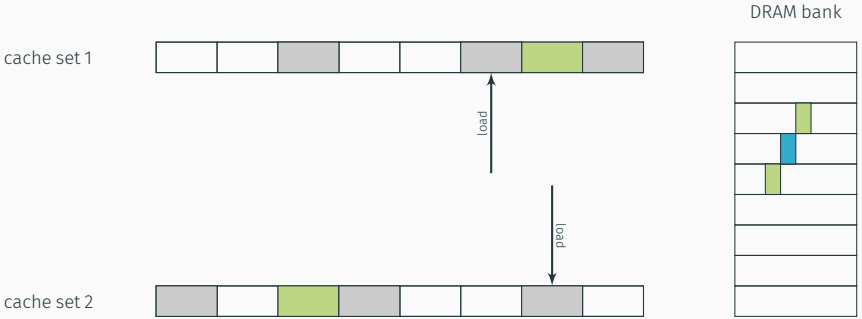
#2 Hammering with cache eviction



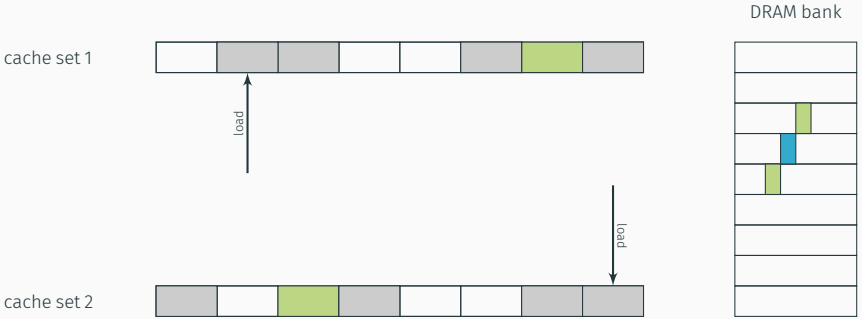
#2 Hammering with cache eviction



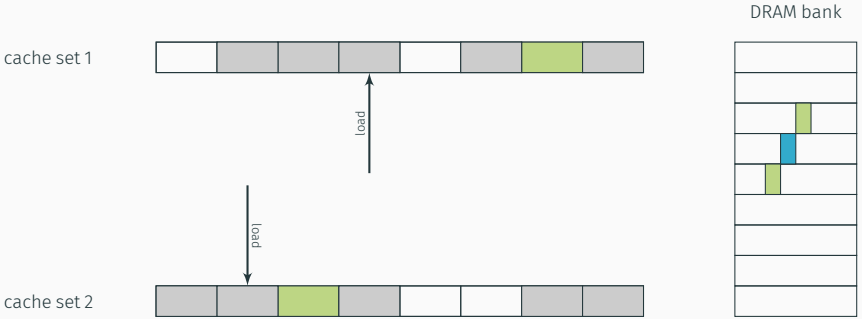
#2 Hammering with cache eviction



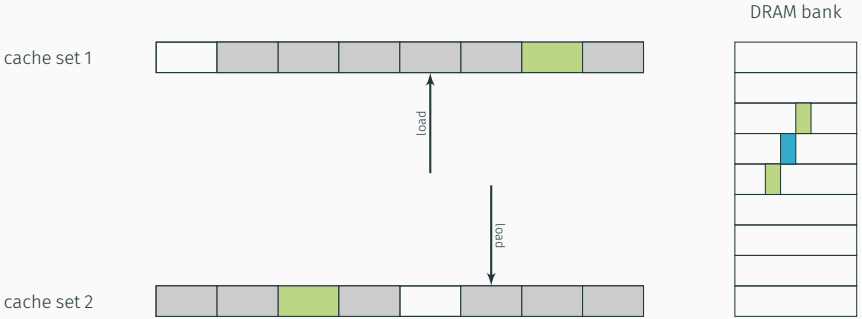
#2 Hammering with cache eviction



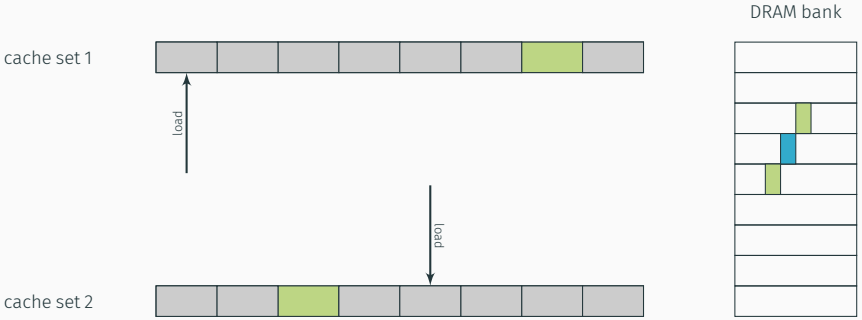
#2 Hammering with cache eviction



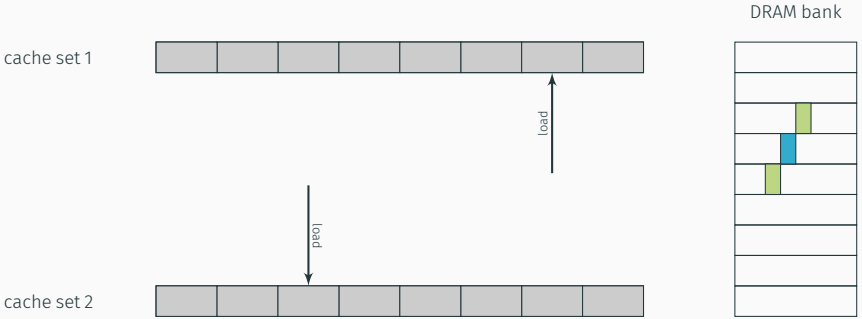
#2 Hammering with cache eviction



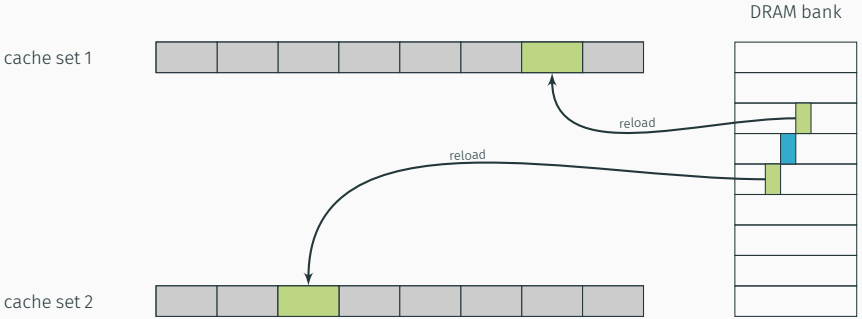
#2 Hammering with cache eviction



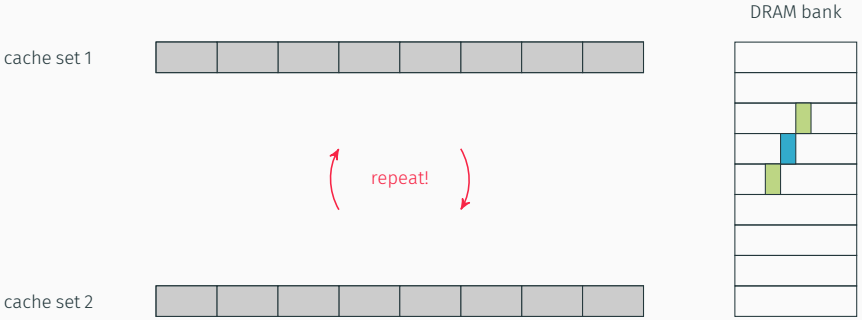
#2 Hammering with cache eviction



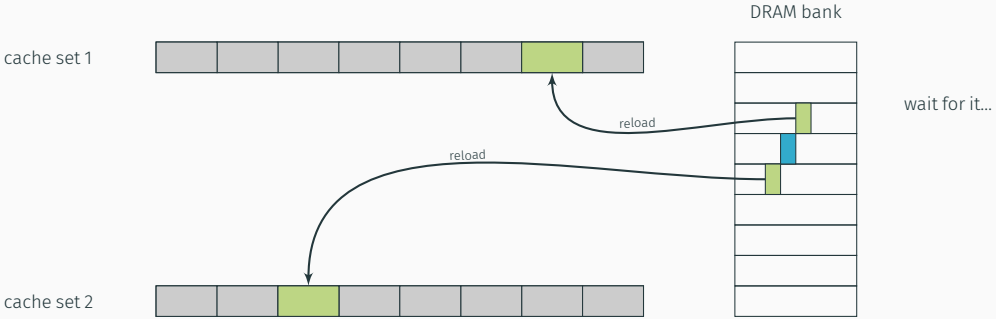
#2 Hammering with cache eviction



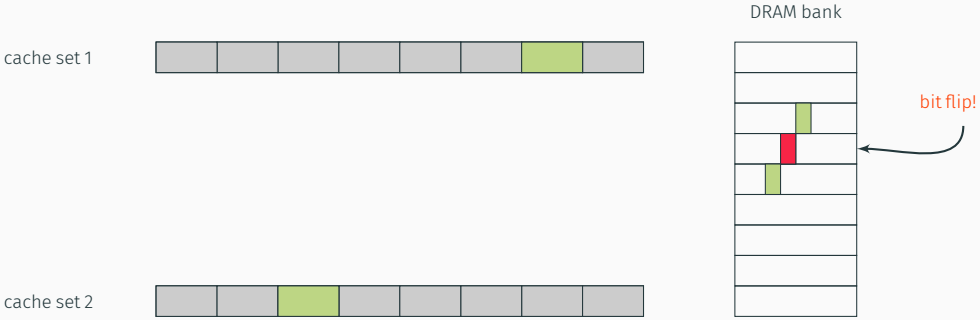
#2 Hammering with cache eviction



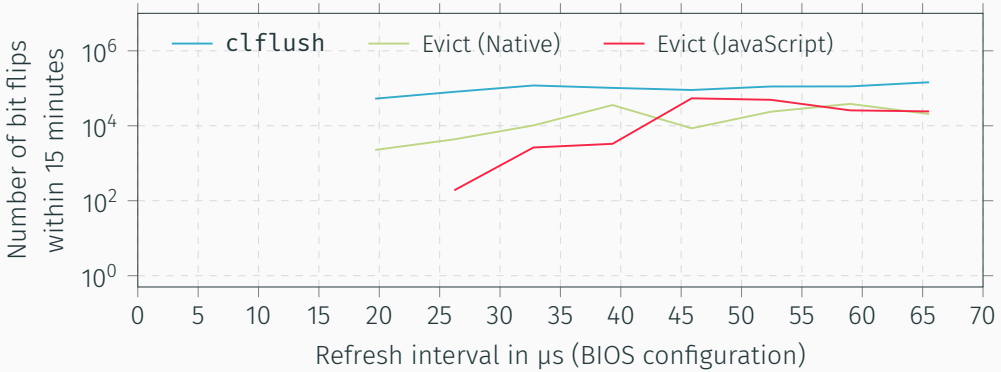
#2 Hammering with cache eviction



#2 Hammering with cache eviction



#2 Hammering with cache eviction: Evaluation on Haswell



#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

Rui Qiao et al. “A new approach for rowhammer attacks”. In: *HOST 2016*. 2016.

#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
- issue: NT stores to 1 address are combined at write-combining buffer
- only last write goes to DRAM $\rightarrow$ rate not sufficient

Rui Qiao et al. "A new approach for rowhammer attacks". In: *HOST 2016*. 2016.

#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
- issue: NT stores to 1 address are combined at write-combining buffer
- only last write goes to DRAM $\rightarrow$ rate not sufficient
- solution: following cached access to same address

Rui Qiao et al. "A new approach for rowhammer attacks". In: *HOST 2016*. 2016.

#3 Hammering with non-temporal accesses

```
begin:
  movnti %eax, (X)
  movnti %eax, (Y)

  jmp begin
```

→

```
begin:
  movnti %eax, (X)
  movnti %eax, (Y)
  mov %eax, (X)
  mov %eax, (Y)
  jmp begin
```

#4 Hammering with uncached memory

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

Victor van der Veen et al. “Drammer: Deterministic Rowhammer Attacks on Mobile Platforms”. In: *CCS'16*. 2016.

#4 Hammering with uncached memory

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

- ARMv7 flush instruction is privileged

Victor van der Veen et al. “Drammer: Deterministic Rowhammer Attacks on Mobile Platforms”. In: *CCS'16*. 2016.

#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

Victor van der Veen et al. “Drammer: Deterministic Rowhammer Attacks on Mobile Platforms”. In: *CCS'16*. 2016.

#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

Victor van der Veen et al. “Drammer: Deterministic Rowhammer Attacks on Mobile Platforms”. In: *CCS'16*. 2016.

#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

#5 Remote hammering

- previous work: some code execution (even JS)
- how about remote attacks, *i.e.*, triggered by network packets?
  - Tatar et al. use RDMA, fast network communication that does not involve the CPU
  - Lipp et al. use Intel CAT that restricts cache allocation to a subset of cache ways for QoS

Moritz Lipp et al. “Nethammer: Inducing Rowhammer Faults through Network Requests”. In: *arXiv:1805.04956* (2018).

Andrei Tatar et al. “Throwhammer: Rowhammer Attacks over the Network and Defenses”. In: *USENIX ATC 2018*. 2018.

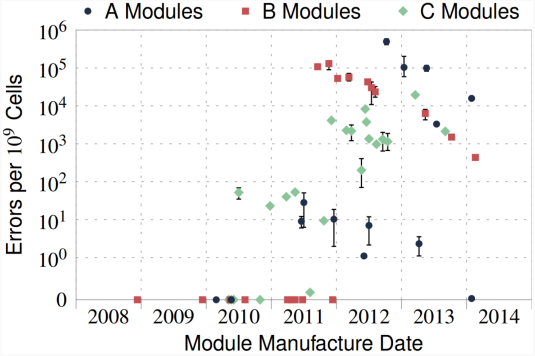
How widespread is the issue?

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:

- still bit flips (Pessl et al.)



Prevalence, by Kim et al.

Peter Pessl et al. "DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks". In: *USENIX Security Symposium*. 2016.

Yoongu Kim et al. "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors". In: *ISCA'14*. 2014.

Issue #2: How do we target accesses?

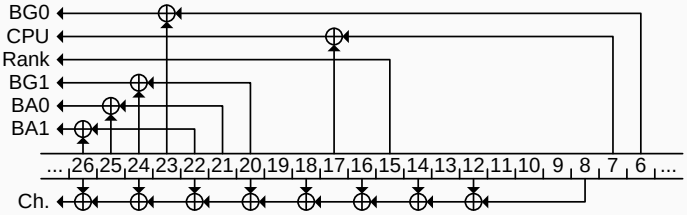
Physical addresses and DRAM

- fixed map: physical addresses → DRAM cells
- undocumented for Intel

Peter Pessl et al. “DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks”. In: *USENIX Security Symposium*. 2016.

Physical addresses and DRAM

- fixed map: physical addresses → DRAM cells
- undocumented for Intel → **reverse-engineered by Pessl et al.**



Peter Pessl et al. “DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks”. In: *USENIX Security Symposium*. 2016.

Memory controller policies

- open-page policy: keep row opened and buffered
  - low latency for subsequent accesses to same row
  - high latency for accesses to any other row

Daniel Gruss et al. “Another Flip in the Wall of Rowhammer Defenses”. In: *S&P'18*. 2018.

Memory controller policies

- open-page policy: keep row opened and buffered
  - low latency for subsequent accesses to same row
  - high latency for accesses to any other row

- close-page policy: immediately close row, ready to open a new row
  - medium latency for accesses to any row
  - perform better on multi-core systems

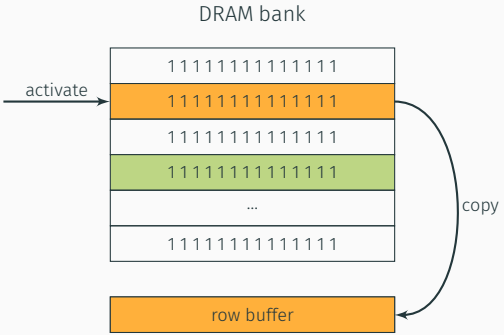
Daniel Gruss et al. "Another Flip in the Wall of Rowhammer Defenses". In: *S&P'18*. 2018.

“Double-sided hammering”

With an open-page policy, you need to alternate accesses to two rows

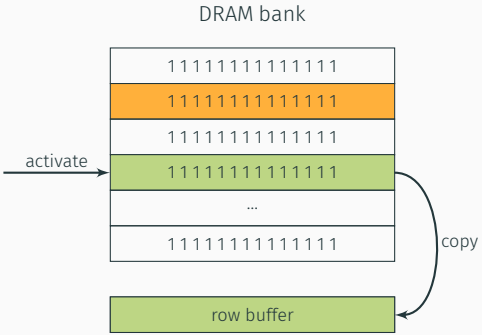
“Double-sided hammering”

With an open-page policy, you need to alternate accesses to two rows



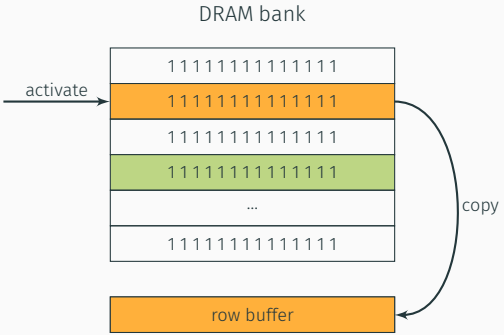
“Double-sided hammering”

With an open-page policy, you need to alternate accesses to two rows



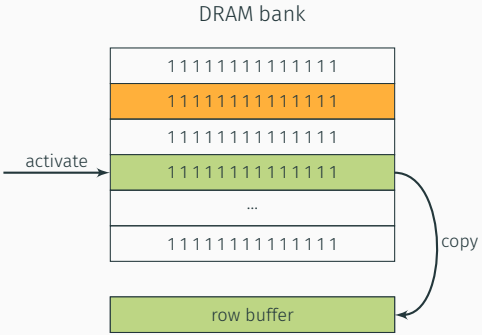
“Double-sided hammering”

With an open-page policy, you need to alternate accesses to two rows



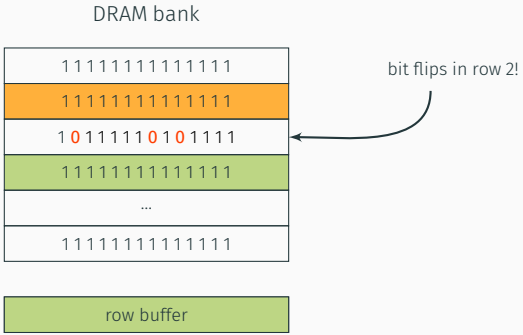
“Double-sided hammering”

With an open-page policy, you need to alternate accesses to two rows



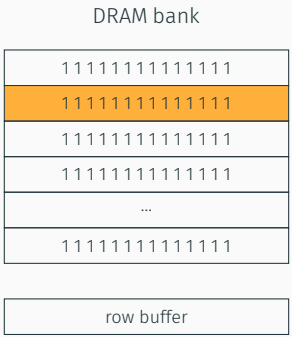
“Double-sided hammering”

With an open-page policy, you need to alternate accesses to two rows



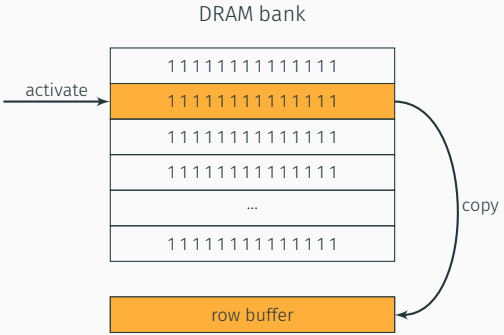
“One-location hammering”

With a close-page policy, you can hammer a single row



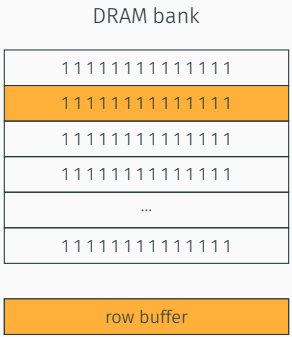
“One-location hammering”

With a close-page policy, you can hammer a single row



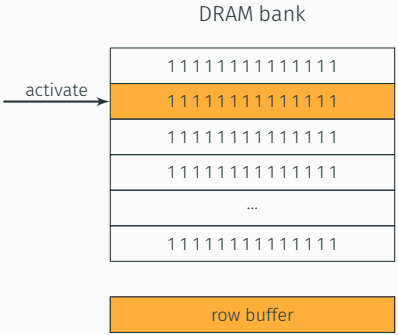
“One-location hammering”

With a close-page policy, you can hammer a single row



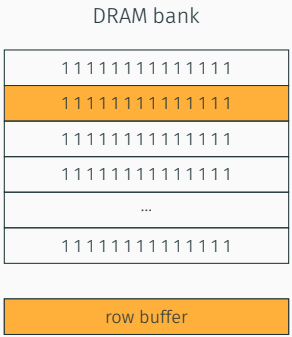
“One-location hammering”

With a close-page policy, you can hammer a single row



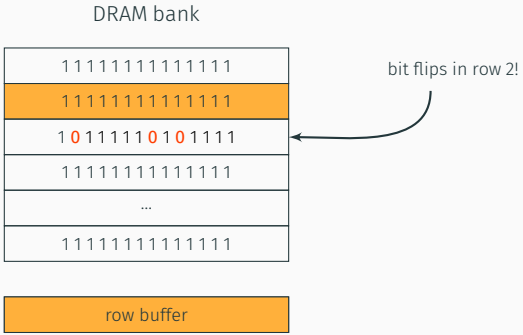
“One-location hammering”

With a close-page policy, you can hammer a single row



“One-location hammering”

With a close-page policy, you can hammer a single row



Issue #3: Can we exploit these bit flips?

How to exploit random bit flips?

- Rowhammer was deemed non-exploitable and only a reliability issue

How to exploit random bit flips?

- Rowhammer was deemed non-exploitable and only a reliability issue
- bit flips are not random → highly reproducible flip pattern!
- ideas for exploitation
  1. bit flip in data structure, e.g., page table
  2. bit flip in instruction opcode
  3. bit flip in signature (→ fault-based cryptanalysis)

Bit flips in page tables

General idea

1. allocate a large chunk of memory
2. scan for “good” flips with your own buffer
3. return that particular area of memory to the OS
4. force OS to place data structure there → page tables are good for this
5. trigger bit flip again
6. profit

Mark Seaborn. *Exploiting the DRAM rowhammer bug to gain kernel privileges.* `http://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html`. Retrieved on June 26, 2015. 2015.

Bit flips in page tables

- x86 page tables entries (PTEs) control access to physical memory
- bit flip in a PTE's physical page number can give a process access to a different physical page

Bit flips in page tables

- x86 page tables entries (PTEs) control access to physical memory
- bit flip in a PTE's physical page number can give a process access to a different physical page
- aim of exploit: get access to a page table
  $\rightarrow$ gives access to all of physical memory

Bit flips in page tables

- x86 page tables entries (PTEs) control access to physical memory
- bit flip in a PTE's physical page number can give a process access to a different physical page
- aim of exploit: get access to a page table
  $\rightarrow$ gives access to all of physical memory
- maximize chances that a bit flip is useful by "spraying" physical memory with page tables

Bit flips in page tables

Page table: 4k page containing array of 512 PTEs (64 bits each)

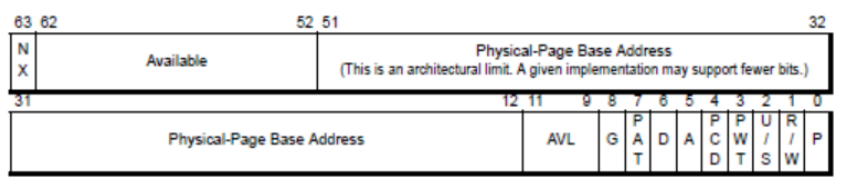


Figure 5-21. 4-Kbyte PTE—Long Mode

Bit flips in page tables

Page table: 4k page containing array of 512 PTEs (64 bits each)

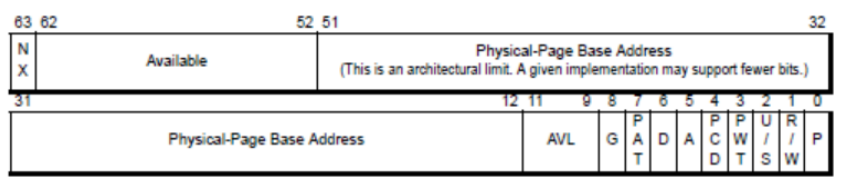


Figure 5-21. 4-Kbyte PTE—Long Mode

Could flip

- "writable" permission bit (RW): 1 bit → 2% chance

Bit flips in page tables

Page table: 4k page containing array of 512 PTEs (64 bits each)

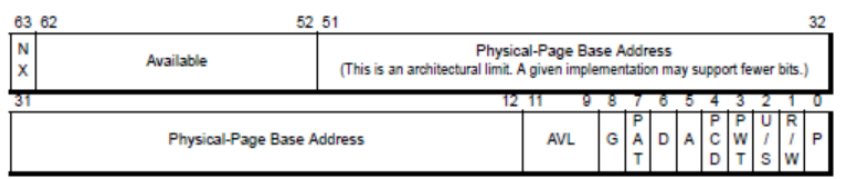
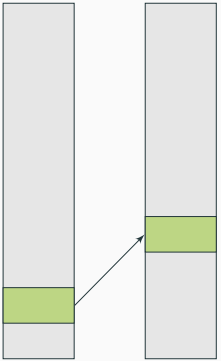


Figure 5-21. 4-Kbyte PTE—Long Mode

Could flip

- "writable" permission bit (RW): 1 bit → 2% chance
- physical page number: 20 bits on 4GB system → 31% chance

Bit flips in page tables

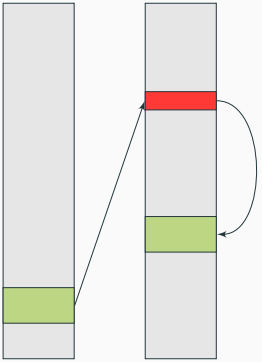


virtual address space

physical memory

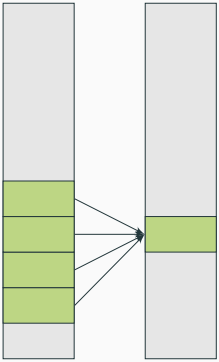
- mapping a file with read-write permissions?

Bit flips in page tables



- mapping a file with read-write permissions?
  $\rightarrow$ indirection via page tables

Bit flips in page tables

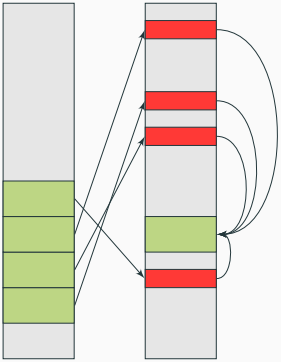


virtual address space

physical memory

- mapping a file with read-write permissions?
  $\rightarrow$ indirection via page tables
- repeatedly mapping a file with read-write permissions?

Bit flips in page tables

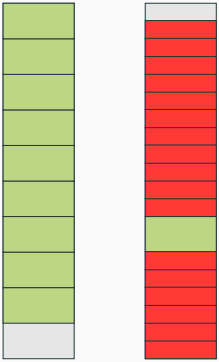


virtual address space

physical memory

- mapping a file with read-write permissions?
  $\rightarrow$ indirection via page tables
- repeatedly mapping a file with read-write permissions?
  $\rightarrow$ more PTEs in physical memory!

Bit flips in page tables

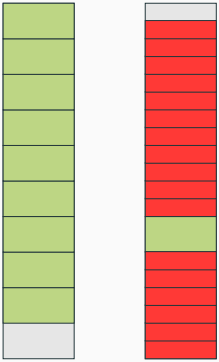


virtual address space

physical memory

- mapping a file with read-write permissions?
  → indirection via page tables
- repeatedly mapping a file with read-write permissions?
  → more PTEs in physical memory!
- we can fill physical memory with PTEs

Bit flips in page tables

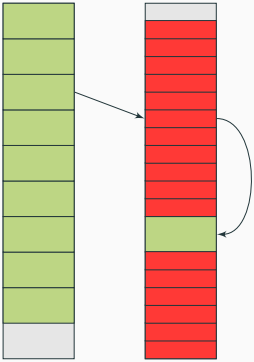


virtual address space

physical memory

- physical memory is filled with PTEs

Bit flips in page tables

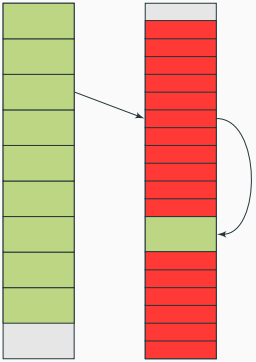


virtual address space

physical memory

- physical memory is filled with PTEs

Bit flips in page tables

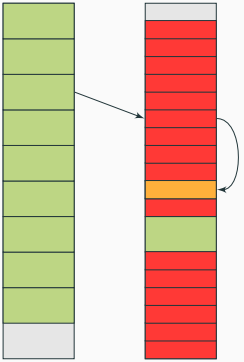


virtual address space

physical memory

- physical memory is filled with PTEs
- if a bit flips in the right place in the PTE...

Bit flips in page tables

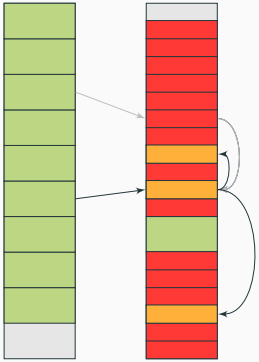


virtual address space

physical memory

- physical memory is filled with PTEs
- if a bit flips in the right place in the PTE...
- ... the corresponding virtual address now points to a wrong physical page, with RW access, with a great chance the page contains a PT itself

Bit flips in page tables

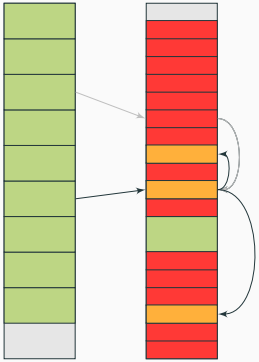


virtual address space

physical memory

- physical memory is filled with PTEs
- if a bit flips in the right place in the PTE...
- ... the corresponding virtual address now points to a wrong physical page, with RW access, with a great chance the page contains a PT itself
- use that to map any memory read/write

Bit flips in page tables



virtual address space

physical memory

- physical memory is filled with PTEs
- if a bit flips in the right place in the PTE...
- ... the corresponding virtual address now points to a wrong physical page, with RW access, with a great chance the page contains a PT itself
- use that to map any memory read/write
- including kernel memory
  $\rightarrow$ privilege escalation

Bit flips in instruction opcode

- some applications perform actions as root
- can be used by unprivileged users

Daniel Gruss et al. “Another Flip in the Wall of Rowhammer Defenses”. In: *S&P'18*. 2018.

Bit flips in instruction opcode

- some applications perform actions as root
- can be used by unprivileged users
- ping, mount, sudo

Daniel Gruss et al. “Another Flip in the Wall of Rowhammer Defenses”. In: *S&P'18*. 2018.

Bit flips in instruction opcode

Bit flips in instruction opcode

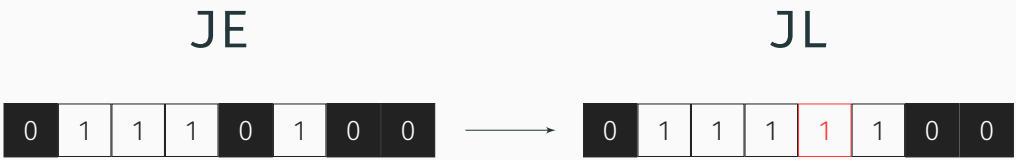
Bit flips in instruction opcode

Bit flips in instruction opcode

JE

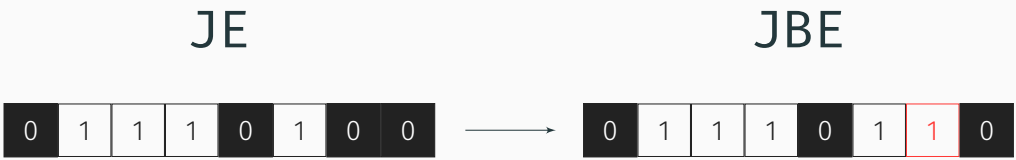
<prefix>

Bit flips in instruction opcode

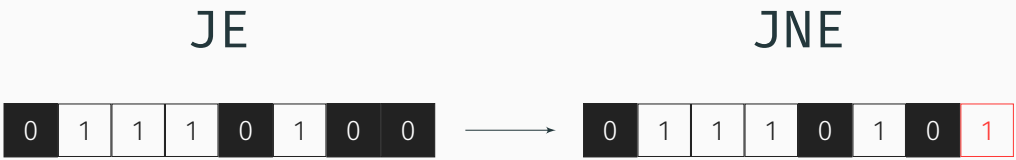


Bit flips in instruction opcode

Bit flips in instruction opcode



Bit flips in instruction opcode



- bit flip in conditional jump → bypass password check

Bit flips in instruction opcode

- not just conditional jump
- other targets include:
  - comparisons
  - addresses of memory loads/stores
- analysis of sudo $\rightarrow$ 29 possible bit flips to bypass password check

Countermeasures

Countermeasures

Different countermeasures have been proposed:

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

Quick fixes

1. no `clflush` instruction

Quick fixes

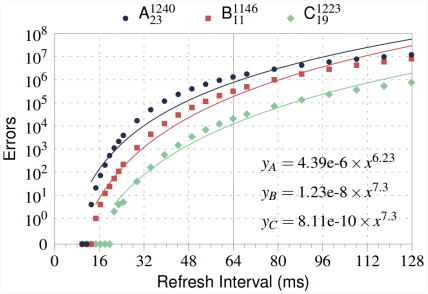
1. no `clflush` instruction → Rowhammer.js

Quick fixes

1. no `clflush` instruction → Rowhammer.js
2. increase the refresh rate

Quick fixes

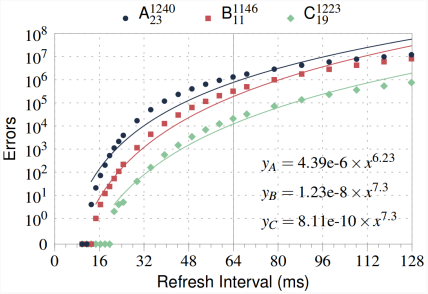
1. no `clflush` instruction → Rowhammer.js
2. increase the refresh rate
   → would need to be increased by 7× to eliminate all bit flips



Errors depending on refresh interval ([Kim+14])

Quick fixes

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



Errors depending on refresh interval ([Kim+14])

ECC

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

Mark Lanteigne. *How Rowhammer Could Be Used to Exploit Weaknesses in Computer Hardware*. 2016. URL: http://www.thirdio.com/rowhammer.pdf.

ECC

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

Mark Lanteigne. *How Rowhammer Could Be Used to Exploit Weaknesses in Computer Hardware*. 2016. URL: http://www.thirdio.com/rowhammer.pdf.

ECC

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

Mark Lanteigne. *How Rowhammer Could Be Used to Exploit Weaknesses in Computer Hardware*. 2016. URL: http://www.thirdio.com/rowhammer.pdf.

ECC

- ECC protection: server can handle or correct single bit errors
- no standard for event reporting
- in practice
  - 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

Mark Lanteigne. *How Rowhammer Could Be Used to Exploit Weaknesses in Computer Hardware*. 2016. URL: http://www.thirdio.com/rowhammer.pdf.

ECC

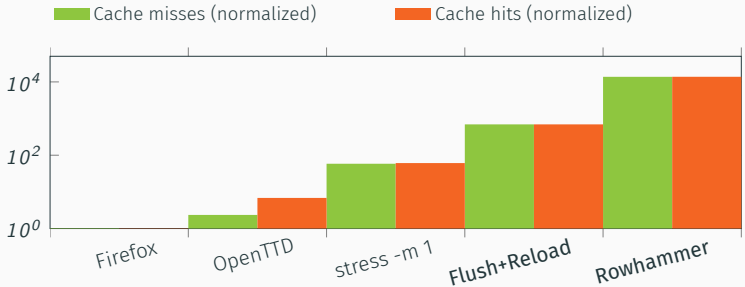
- ECC protection: server can handle or correct single bit errors
- no standard for event reporting
- in practice
  - common: server counts ECC errors and report only if they reach a threshold (e.g., > 100 bit flips / hour)
  - some server vendors never report errors to the OS
  - one server did not even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

---

Mark Lanteigne. *How Rowhammer Could Be Used to Exploit Weaknesses in Computer Hardware*. 2016. URL: http://www.thirdio.com/rowhammer.pdf.

Detecting Rowhammer attacks

- Rowhammer: lots of cache misses that can be monitored with hardware performance counters ([Her+15; Gru+16a; Chi+15; Pay16])



Preventing Rowhammer attacks in hardware (1/3)

Original ideas from [Kim+14]

- 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+14]

- 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

$\rightarrow$ expensive, performance overhead, or increased power consumption

Preventing Rowhammer attacks in hardware (2/3)

PARA - Probabilistic Adjacent Row Activation ([Kim+14])

- one row closed $\rightarrow$ one adjacent row opened with low probability p

Preventing Rowhammer attacks in hardware (2/3)

PARA - Probabilistic Adjacent Row Activation ([Kim+14])

- 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}

Preventing Rowhammer attacks in hardware (2/3)

PARA - Probabilistic Adjacent Row Activation ([Kim+14])

- 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+14])

- 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+14])

- 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

Preventing Rowhammer attacks in hardware (2/3)

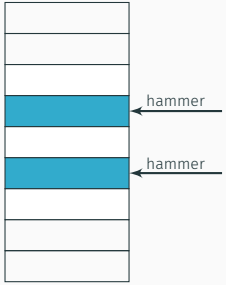
PARA - Probabilistic Adjacent Row Activation ([Kim+14])

- 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 9.4×10^{-14}

Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

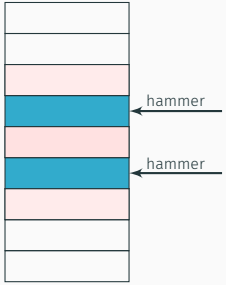
- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold



Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

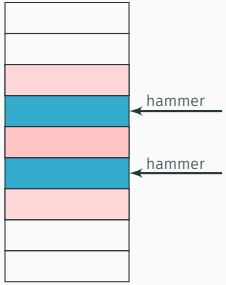
- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold



Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

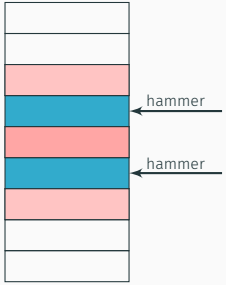
- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold



Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

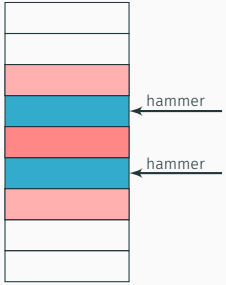
- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold



Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

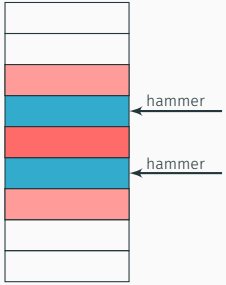
- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold



Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

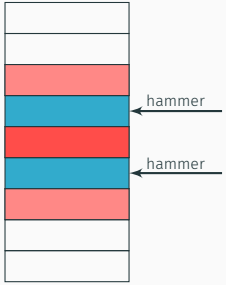
- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold



Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

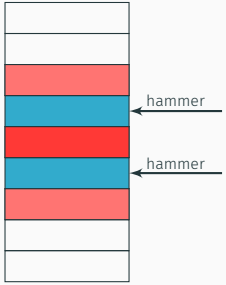
- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold



Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

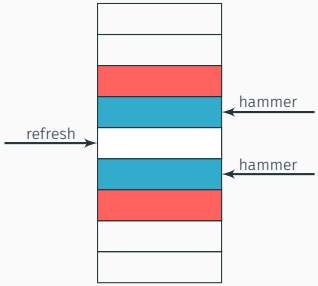
- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold



Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

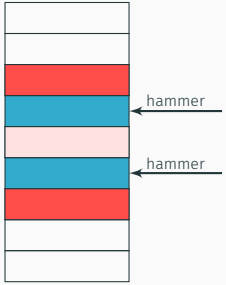
- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold



Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

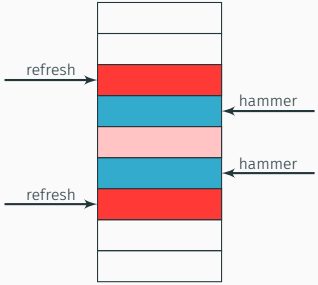
- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold



Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

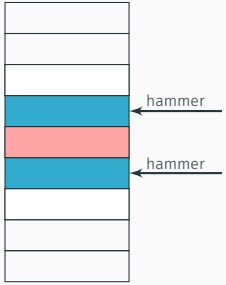
- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold



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 software (1/3)

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

- static analysis of the binary
- detect suspicious instruction sequences (`clflush`, `rdtsc`, fences, …)
- open problem: false positives
- since then: remote exploits from network (Lipp et al., Tatar et al.)

Gorka Irazoqui et al. "MASCAT: Stopping Microarchitectural Attacks Before Execution". In: *Cryptology ePrint Archive: Report 2016/1196* (2016).

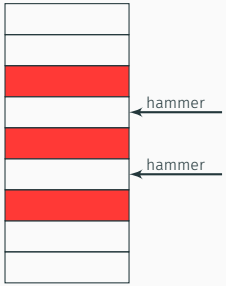
Moritz Lipp et al. "Nethammer: Inducing Rowhammer Faults through Network Requests". In: *arXiv:1805.04956* (2018).

Andrei Tatar et al. "Throwhammer: Rowhammer Attacks over the Network and Defenses". In: *USENIX ATC 2018*. 2018.

Preventing Rowhammer attacks in software (2/3)

ANVIL

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

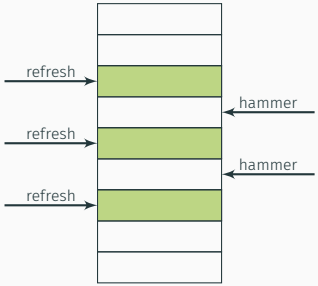


Zelalem Birhanu Aweke et al. “ANVIL: Software-based protection against next-generation rowhammer attacks”. In: *ACM SIGPLAN Notices* 51.4 (2016), pp. 743–755.

Preventing Rowhammer attacks in software (2/3)

ANVIL

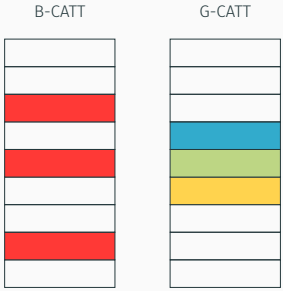
- uses performance counters to detect rowhammer
- activate rows neighbor rows to prevent flips
- similar as PARA, but in software



Zelalem Birhanu Aweke et al. “ANVIL: Software-based protection against next-generation rowhammer attacks”. In: *ACM SIGPLAN Notices* 51.4 (2016), pp. 743–755.

Preventing Rowhammer attacks in software (3/3)

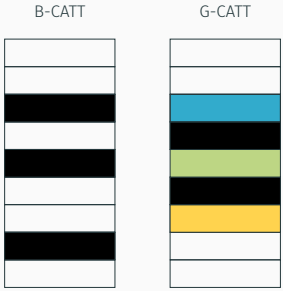
- B-CATT: disable vulnerable physical memory
- G-CATT: isolate security domains in physical memory based on potential vulnerability



Ferdinand Brasser et al. “CAn’t Touch This: Practical and Generic Software-only Defenses Against Rowhammer Attacks”. In: *arXiv:1611.08396* (2016).

Preventing Rowhammer attacks in software (3/3)

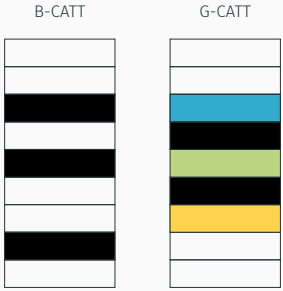
- B-CATT: disable vulnerable physical memory
- G-CATT: isolate security domains in physical memory based on potential vulnerability



Ferdinand Brasser et al. “CAn’t Touch This: Practical and Generic Software-only Defenses Against Rowhammer Attacks”. In: *arXiv:1611.08396* (2016).

Preventing Rowhammer attacks in software (3/3)

- B-CATT: disable vulnerable physical memory
- G-CATT: isolate security domains in physical memory based on potential vulnerability



B-CATT: might block 95% of RAM
G-CATT: what about non-kernel or shared pages?
G-CATT: bit flips more than 8 “rows” apart

Ferdinand Brasser et al. “CAn’t Touch This: Practical and Generic Software-only Defenses Against Rowhammer Attacks”. In: *arXiv:1611.08396* (2016).

Conclusion

- software fault attacks are real, and can be triggered by various techniques in various environments
- no physical access $\rightarrow$ different countermeasures than physical fault attacks
- difficult to replace hardware at a large scale $\rightarrow$ software countermeasures are our best short-term/mid-term hope

Thank you!

Contact

- clementine.maurice@irisa.fr
- @BloodyTangerine

Micro-Architectural Fault Attacks

Clémentine Maurice, CNRS

May 5, 2019—FICHSA, Ben-Gurion University of the Negev, Beer Sheva, Israel

References i

Zelalem Birhanu Aweke, Salessawi Ferede Yitbarek, Rui Qiao, Reetuparna Das, Matthew Hicks, Yossi Oren, and Todd Austin. “ANVIL: Software-based protection against next-generation rowhammer attacks”. In: *ACM SIGPLAN Notices* 51.4 (2016), pp. 743–755.

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

Marco Chiappetta, Erkay Savas, and Cemal Yilmaz. *Real time detection of cache-based side-channel attacks using Hardware Performance Counters*. Cryptology ePrint Archive, Report 2015/1034. 2015.

Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. “Flush+Flush: A Fast and Stealthy Cache Attack”. In: *DIMVA’16*. 2016.

Daniel Gruss, Clémentine Maurice, and Stefan Mangard. “Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript”. In: *DIMVA’16*. 2016.

References ii

Daniel Gruss, Moritz Lipp, Michael Schwarz, Daniel Genkin, Jonas Juffinger, Sioli O'Connell, Wolfgang Schoechl, and Yuval Yarom. "Another Flip in the Wall of Rowhammer Defenses". In: *S&P'18*. 2018.

Nishad Herath and Anders Fogh. "These are Not Your Grand Daddys CPU Performance Counters – CPU Hardware Performance Counters for Security". In: *Black Hat 2015 Briefings*. 2015.

Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. "MASCAT: Stopping Microarchitectural Attacks Before Execution". In: *Cryptology ePrint Archive: Report 2016/1196* (2016).

Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors". In: *ISCA'14*. 2014.

Mark Lanteigne. *How Rowhammer Could Be Used to Exploit Weaknesses in Computer Hardware*. 2016. URL: `http://www.thirdio.com/rowhammer.pdf`.

Moritz Lipp, Misiker Tadesse Aga, Michael Schwarz, Daniel Gruss, Clémentine Maurice, Lukas Raab, and Lukas Lamster. "Nethammer: Inducing Rowhammer Faults through Network Requests". In: *arXiv:1805.04956* (2018).

References iii

Matthias Payer. “HexPADS: a platform to detect “stealth” attacks”. In: *ESSoS’16*. 2016.

Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. “DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks”. In: *USENIX Security Symposium*. 2016.

Rui Qiao and Mark Seaborn. “A new approach for rowhammer attacks”. In: *HOST 2016*. 2016.

Mark Seaborn. *Exploiting the DRAM rowhammer bug to gain kernel privileges*. `http://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html`. Retrieved on June 26, 2015. 2015.

Andrei Tatar, Radhesh Krishnan Konoth, Elias Athanasopoulos, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. “Throwhammer: Rowhammer Attacks over the Network and Defenses”. In: *USENIX ATC 2018*. 2018.

Victor van der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clémentine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. “Drammer: Deterministic Rowhammer Attacks on Mobile Platforms”. In: *CCS’16*. 2016.