Rowhammer: From the Basics to Sophisticated New Variants

Moritz Lipp, Michael Schwarz

February 20, 2018

Graz University of Technology

Who are we

- **Moritz Lipp**
- PhD Student @ Graz University of Technology
- @mlqxyz
- moritz.lipp@iaik.tugraz.at

- **Michael Schwarz**
- PhD Student @ Graz University of Technology
- @misc0110
- michael.schwarz@iaik.tugraz.at

- Security and privacy rely on secrets (unknown to attackers)

- Security and privacy rely on secrets (unknown to attackers)
- Secrets can leak through side channels

Software-based Side-Channel Attacks

- Security and privacy rely on secrets (unknown to attackers)
- Secrets can leak through side channels
- Software-based attacks $\rightarrow$ no physical access

Software-based side-channel attacks leak secrets via...

- Algorithm runtimes
- Cache states
- DRAM states
- ...

- Rowhammer has a special rule among microarchitectural attacks

- Rowhammer has a special rule among microarchitectural attacks
- Does not leak secrets...

Rowhammer

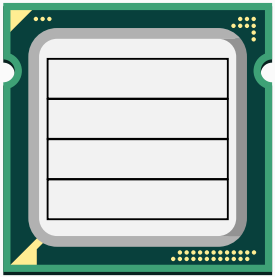
- Rowhammer has a special rule among microarchitectural attacks
- Does not leak secrets...
- ...but it can change memory contents

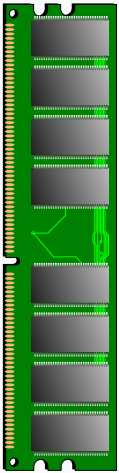
Rowhammer

- Rowhammer has a special rule among microarchitectural attacks
- Does not leak secrets...
- ...but it can change memory contents
- A software-based fault attack on the DRAM

Background

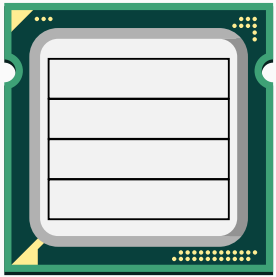
```
printf("%d", i);
printf("%d", i);
```

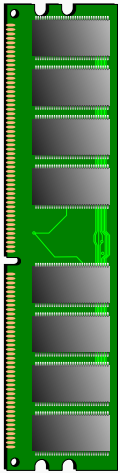




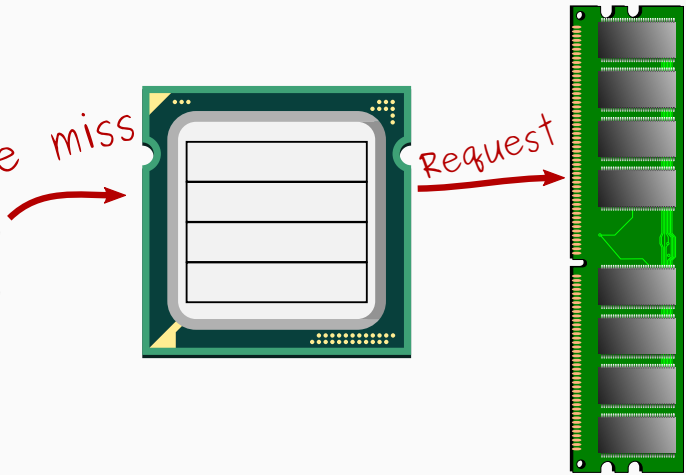
```
printf("%d", i);
printf("%d", i);
```

Cache miss

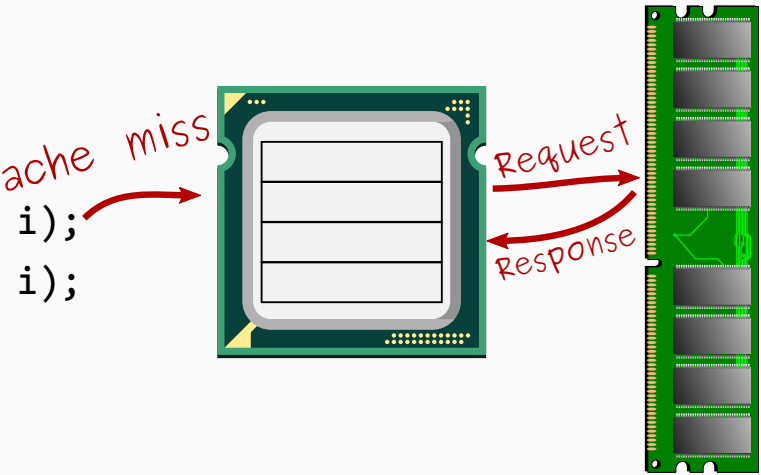




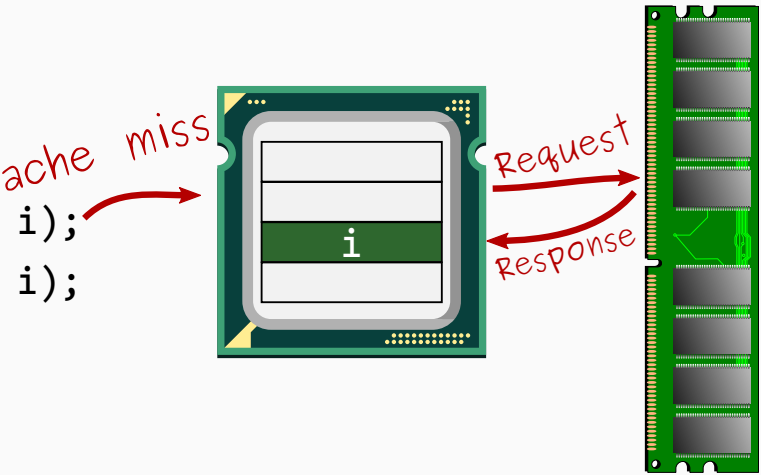
```
printf("%d", i);
printf("%d", i);
```

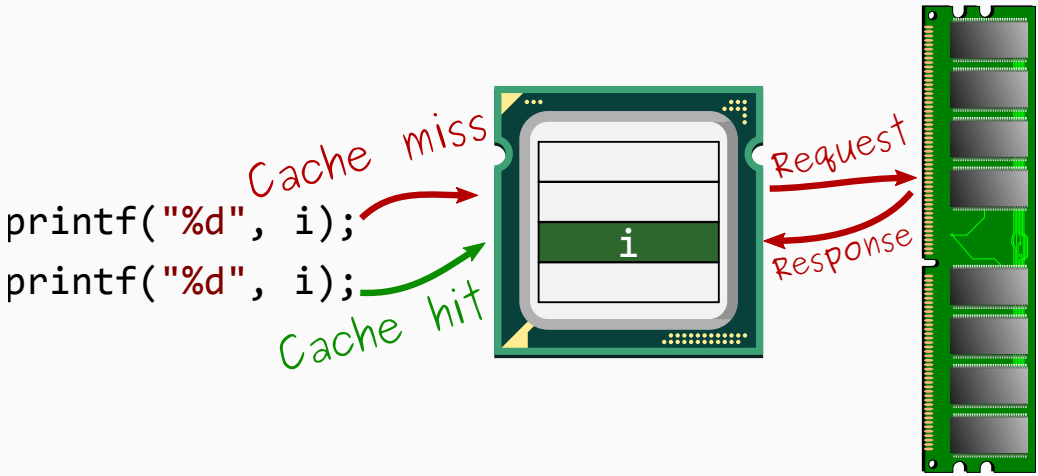


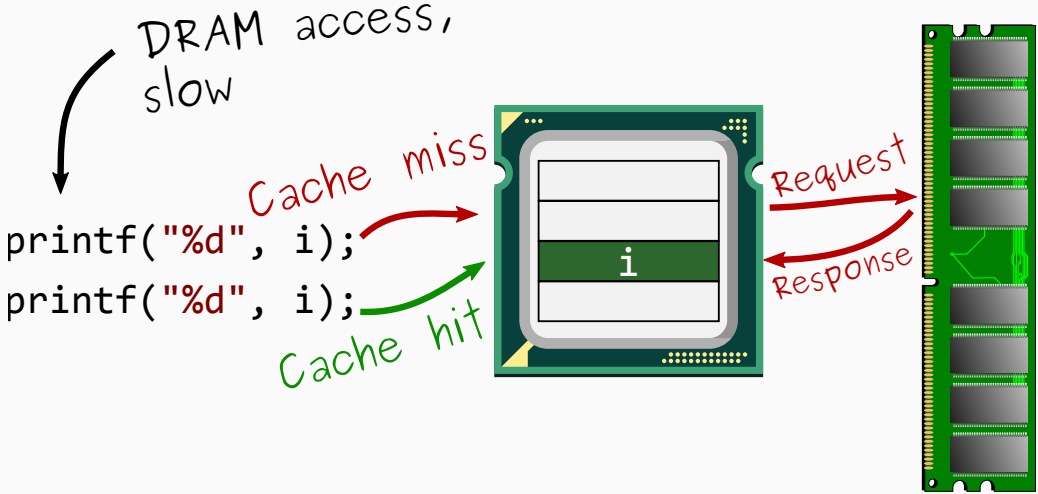
```
printf("%d", i);
printf("%d", i);
```

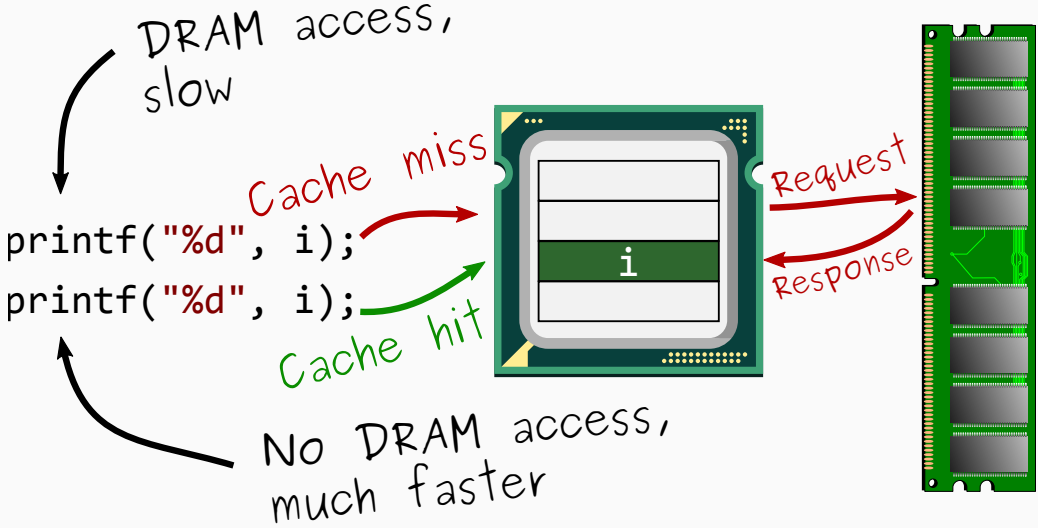


```
printf("%d", i);
printf("%d", i);
```

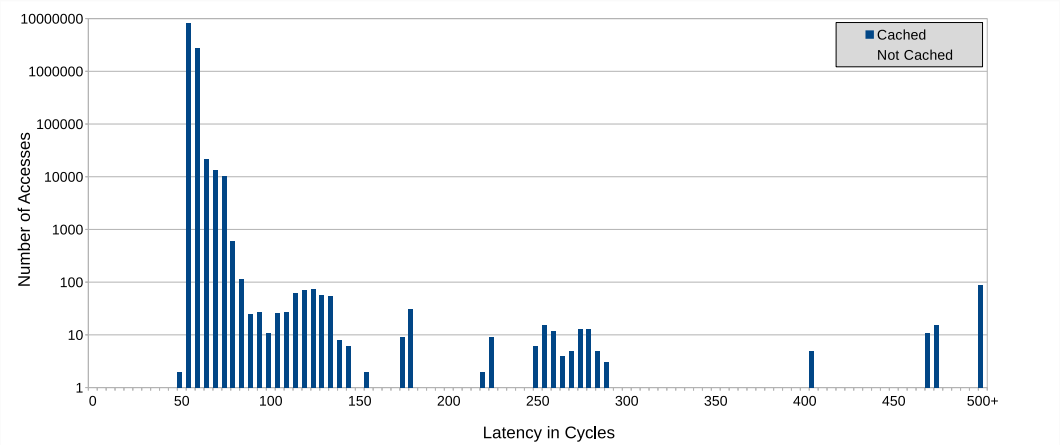



printf("%d", i);
printf("%d", i);
Cache miss
Cache hit
i
Request
Response

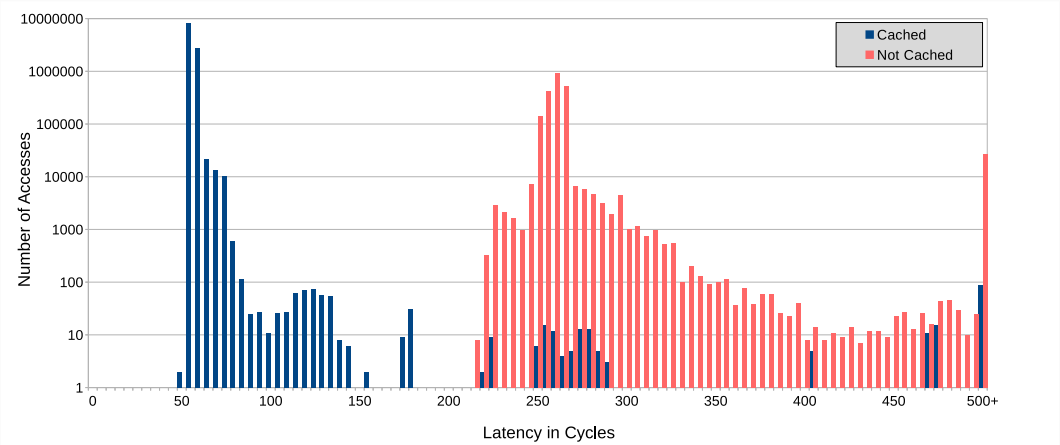

DRAM access, slow
printf("%d", i);
printf("%d", i);
Cache miss
Cache hit
i
Request
Response


DRAM access,
slow
Cache miss
printf("%d", i);
printf("%d", i);
Cache hit
No DRAM access,
much faster
i
Request
Response

Memory Access Latency

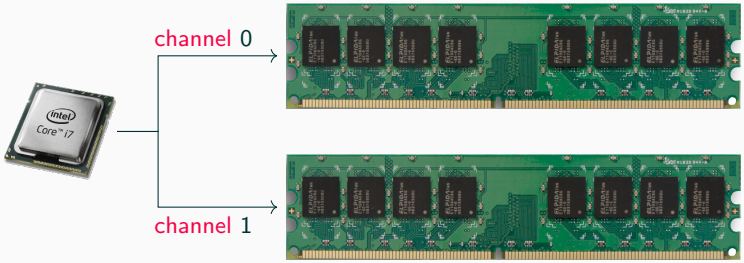


Memory Access Latency

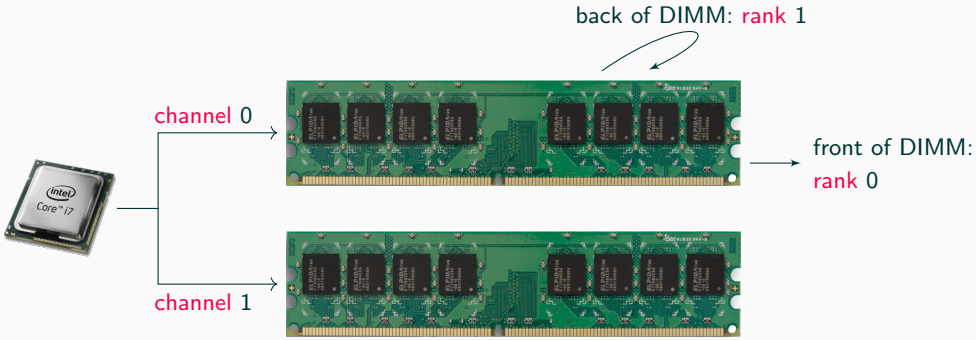


DRAM organization

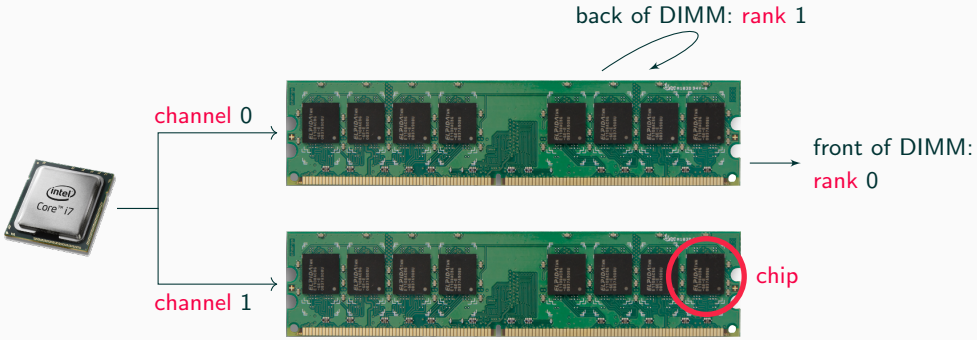
DRAM organization



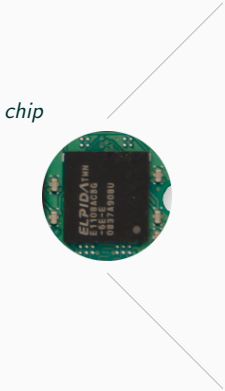
DRAM organization



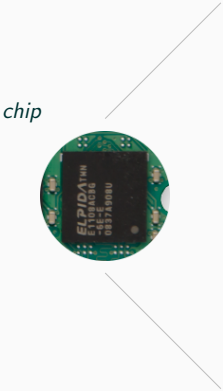
DRAM organization



DRAM organization



DRAM organization



The row buffer

- DRAM internally is only capable of reading entire rows

The row buffer

- DRAM internally is only capable of reading entire rows
- Capacitors in cells discharge when reading them

The row buffer

- DRAM internally is only capable of reading entire rows
- Capacitors in cells discharge when reading them
- Bits are buffered when reading them from the cells

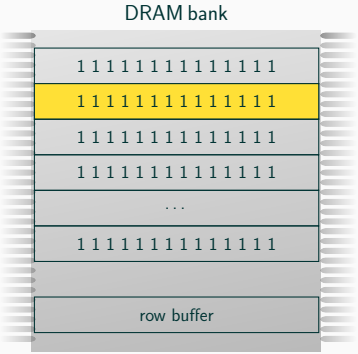
The row buffer

- DRAM internally is only capable of reading entire rows
- Capacitors in cells discharge when reading them
- Bits are buffered when reading them from the cells
- Then, bits are written back to the cells again

The row buffer

- DRAM internally is only capable of reading entire rows
- Capacitors in cells discharge when reading them
- Bits are buffered when reading them from the cells
- Then, bits are written back to the cells again
- $\rightarrow$ Row buffer

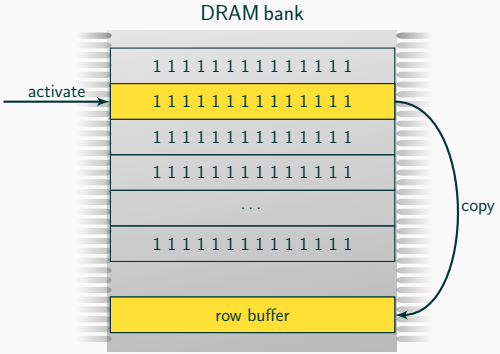
Reading from DRAM



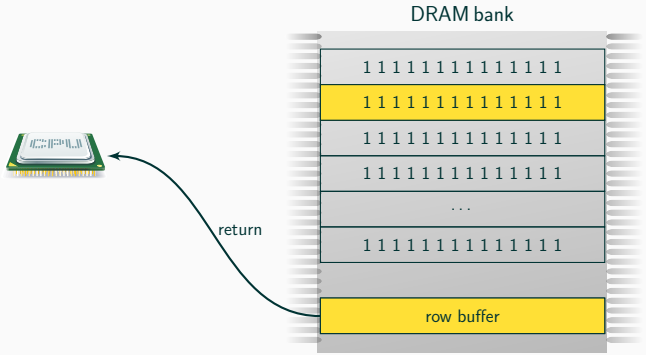
CPU reads row 1:

$\rightarrow$ activate and copy to row buffer

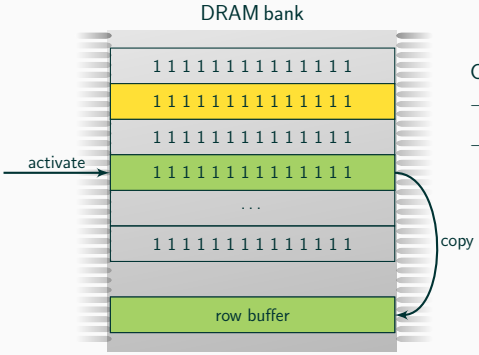
Reading from DRAM



Reading from DRAM



Reading from DRAM

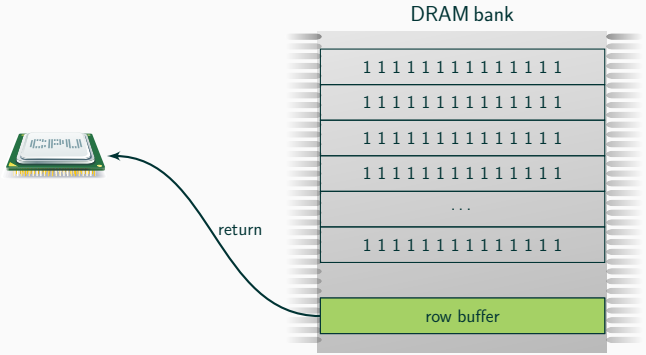


CPU reads row 2:

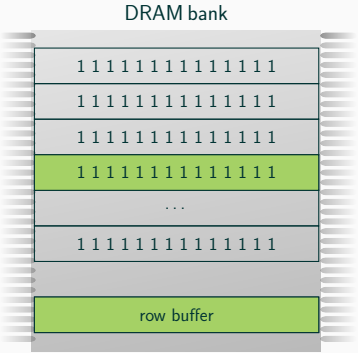
→ activate and copy to row buffer

→ row conflict → slow

Reading from DRAM



Reading from DRAM

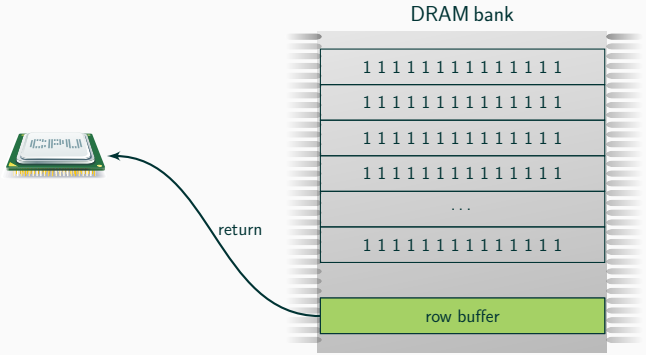


CPU reads row 2:

$\rightarrow$ already in row buffer

$\rightarrow$ row hit $\rightarrow$ fast

Reading from DRAM

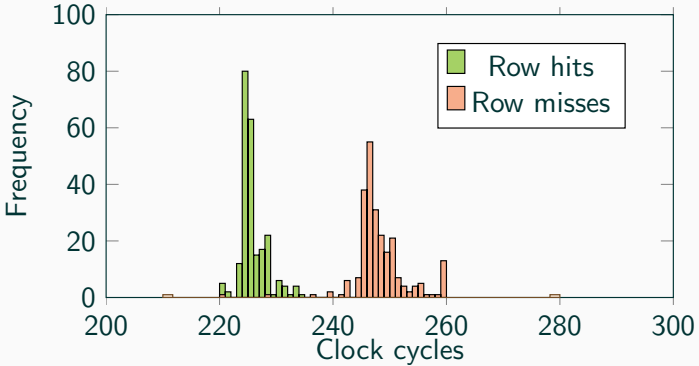


CPU reads row 2:

→ already in row buffer

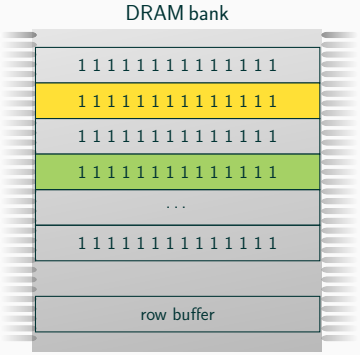
→ row hit → fast

Timing difference

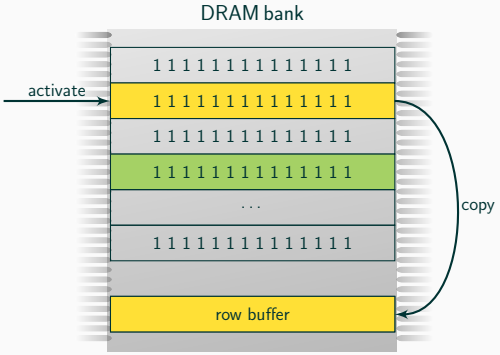


Row hits ($\approx$ 225 cycles) and row conflicts ($\approx$ 247 cycles)

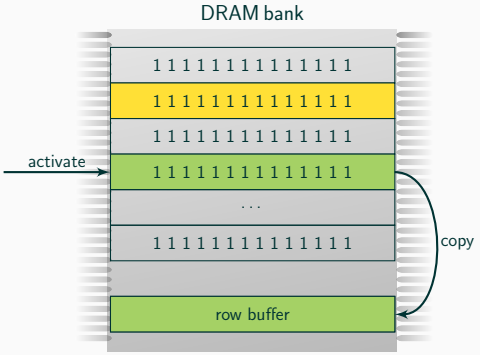
Rowhammer



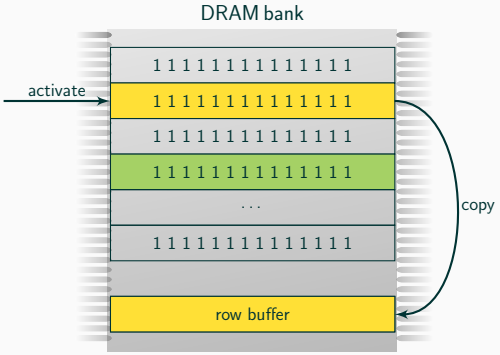
- Cells leak → repetitive refresh necessary
- Maximum interval between refreshes to guarantee data integrity
- Cells leak faster upon proximate accesses → Rowhammer



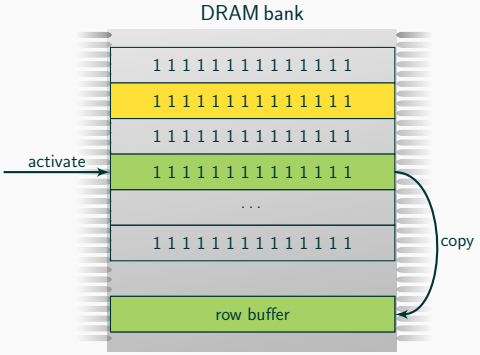
- Cells leak → repetitive refresh necessary
- Maximum interval between refreshes to guarantee data integrity
- Cells leak faster upon proximate accesses → Rowhammer



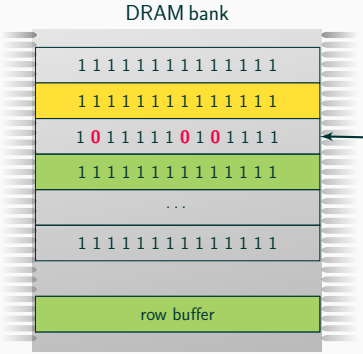
- Cells leak → repetitive refresh necessary
- Maximum interval between refreshes to guarantee data integrity
- Cells leak faster upon proximate accesses → Rowhammer



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



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



bit flips in row 2!

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

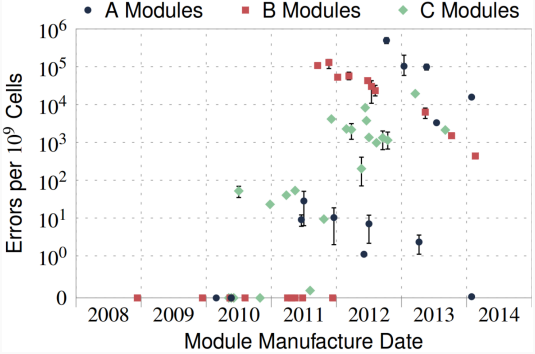
How widespread is the issue?

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)

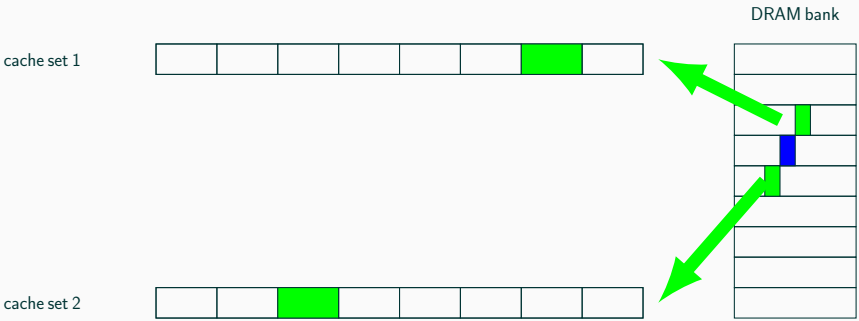


Prevalence, by Kim et al. 2014

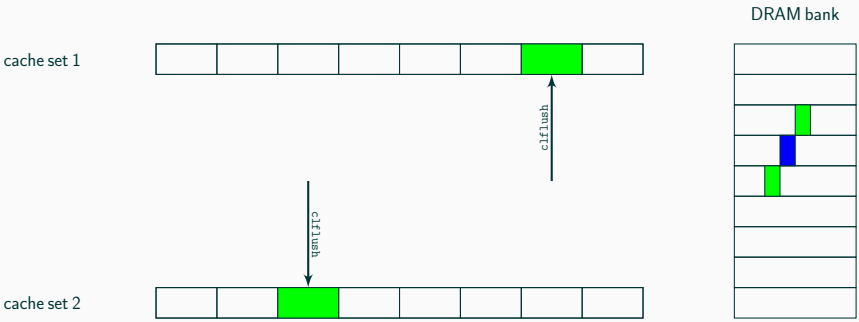
Access techniques

1. Flush cache via `clflush` instruction → original paper (Kim et al. 2014)
2. Cache eviction (Gruss, Maurice, et al. 2016; Aweke et al. 2016)
3. Non-temporal accesses (Qiao et al. 2016)
4. Uncached memory (Veen et al. 2016)

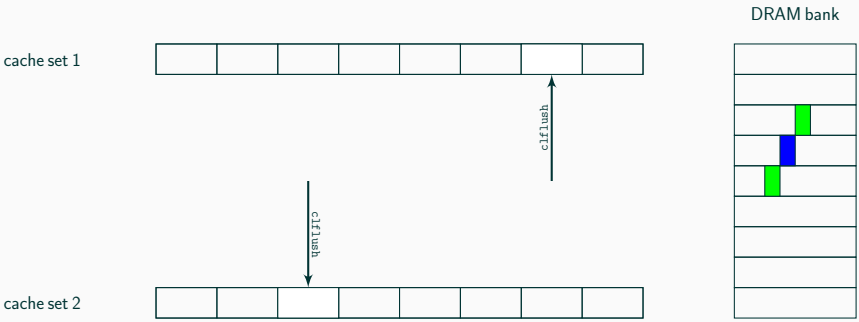
#1 Hammering with `clflush`



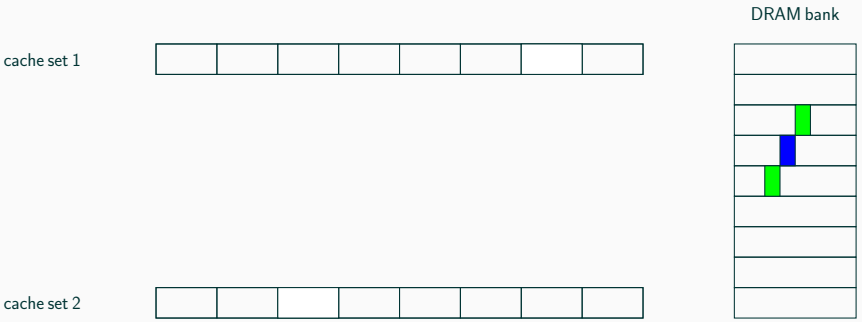
#1 Hammering with `clflush`



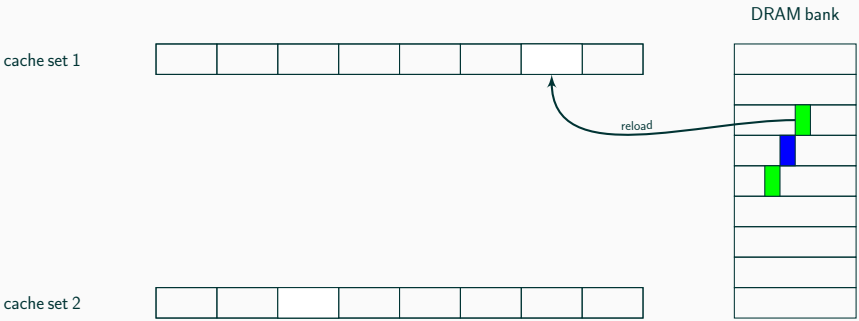
#1 Hammering with `clflush`



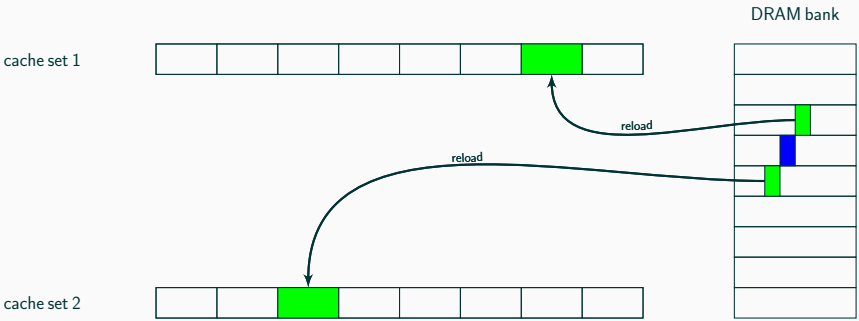
#1 Hammering with `clflush`



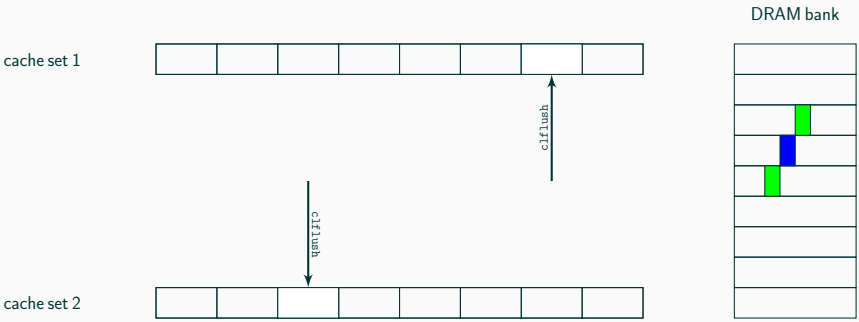
#1 Hammering with `clflush`



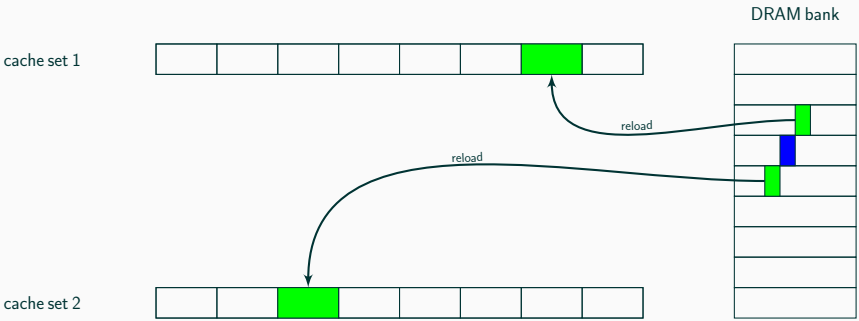
#1 Hammering with `clflush`



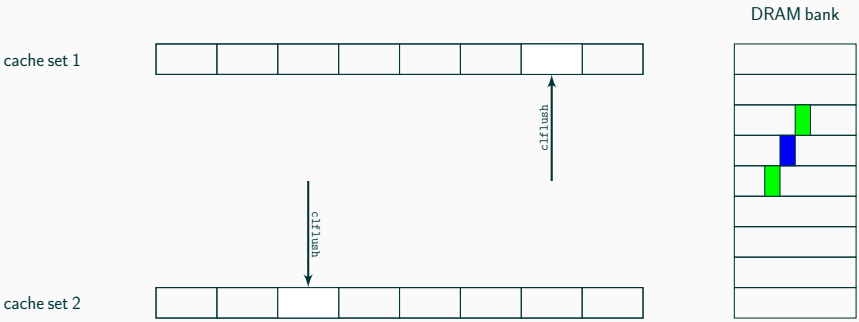
#1 Hammering with `clflush`



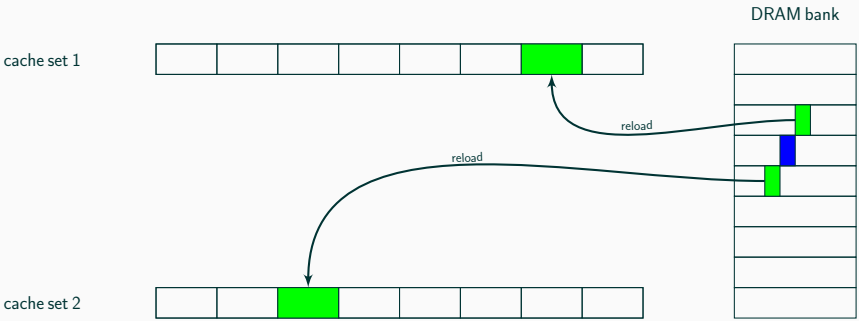
#1 Hammering with `clflush`



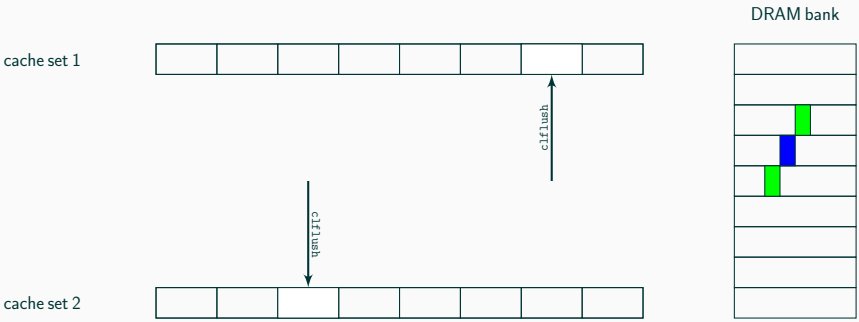
#1 Hammering with `clflush`



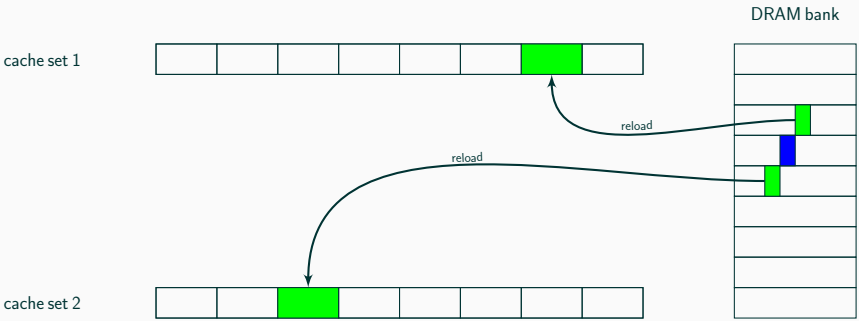
#1 Hammering with `clflush`



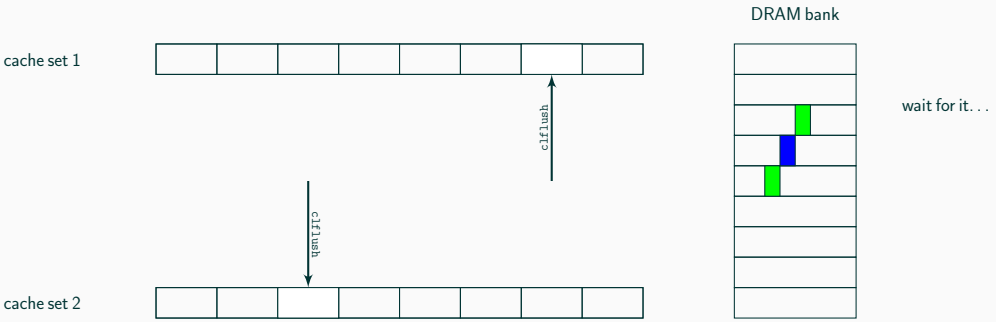
#1 Hammering with `clflush`



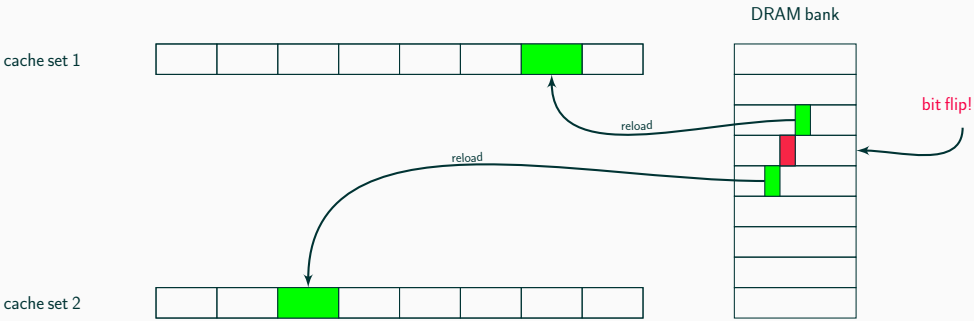
#1 Hammering with `clflush`



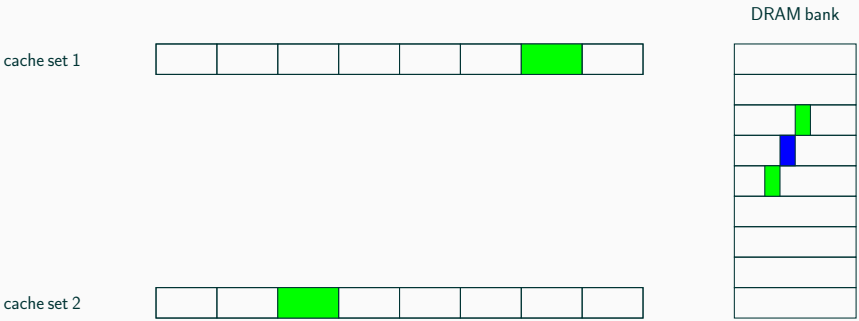
#1 Hammering with `clflush`



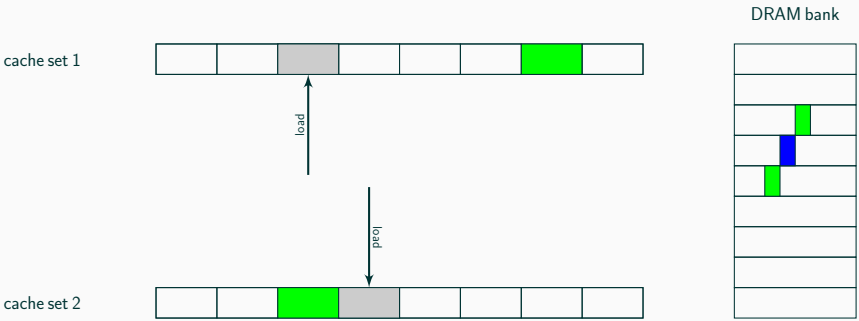
#1 Hammering with `clflush`



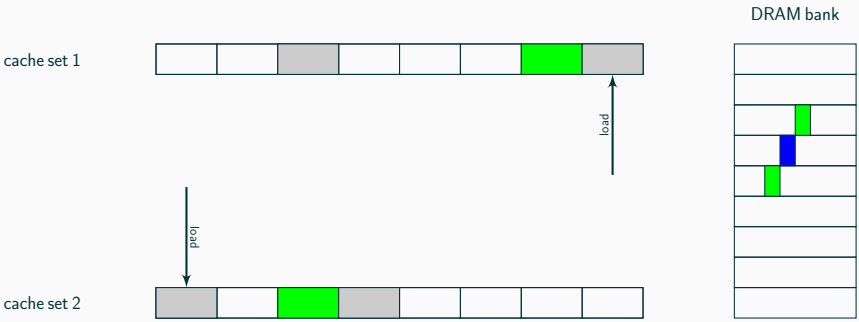
#2 Hammering with cache eviction



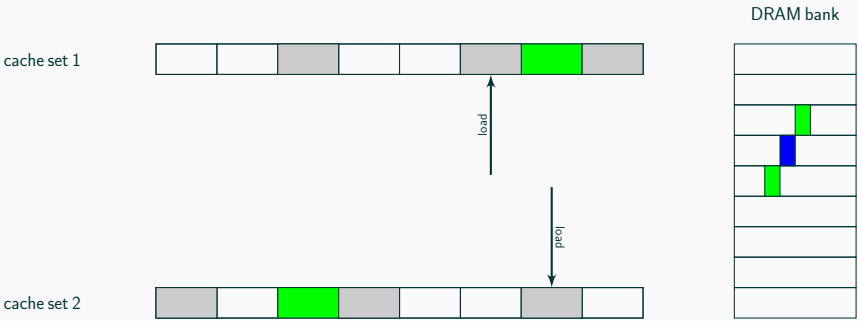
#2 Hammering with cache eviction



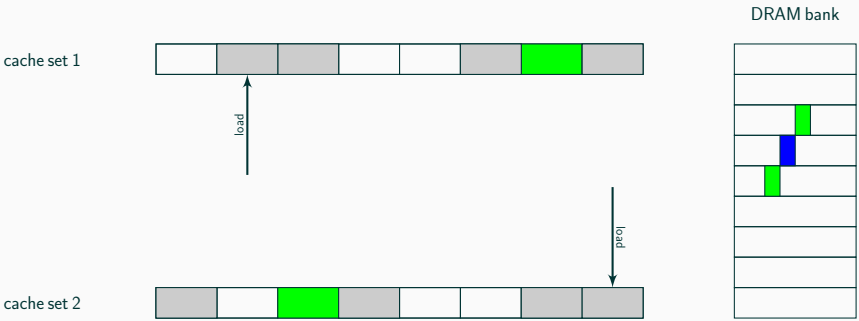
#2 Hammering with cache eviction



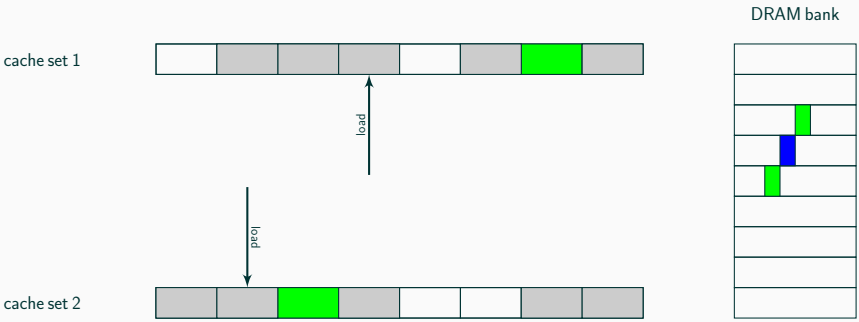
#2 Hammering with cache eviction



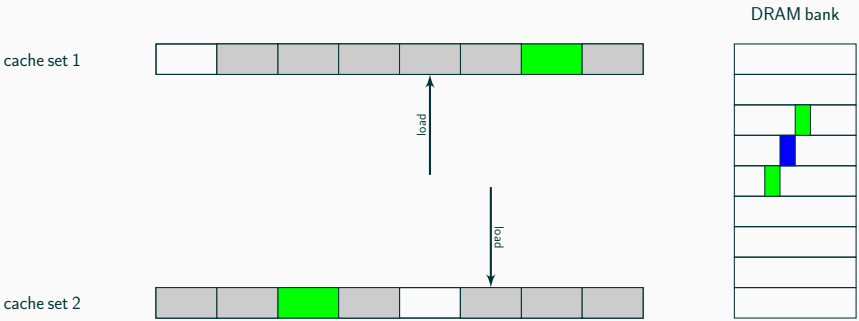
#2 Hammering with cache eviction



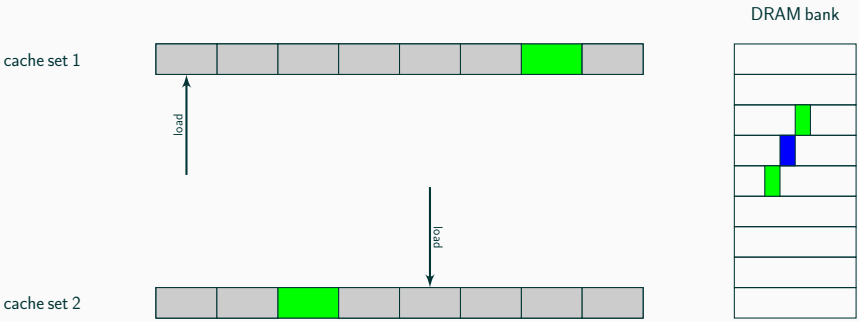
#2 Hammering with cache eviction



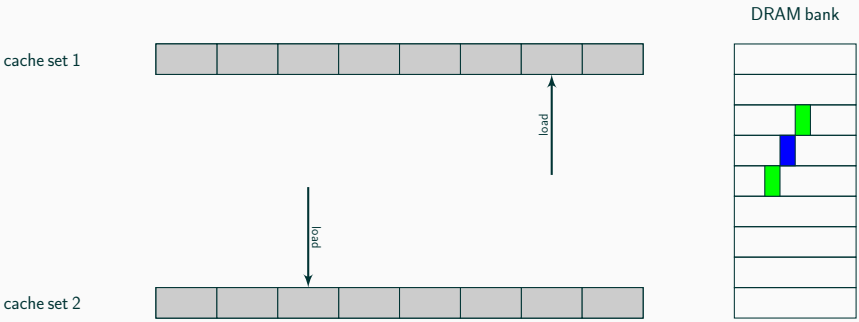
#2 Hammering with cache eviction



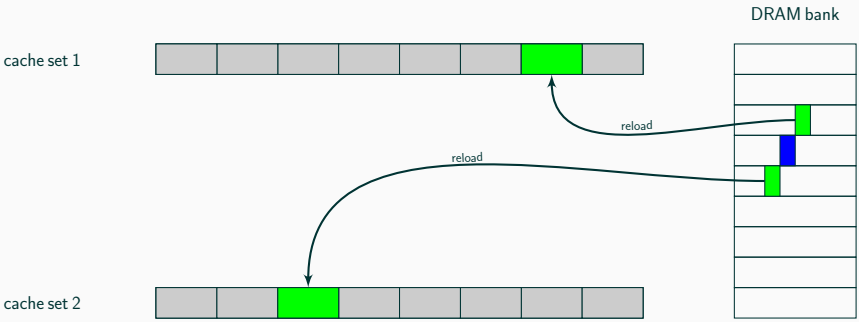
#2 Hammering with cache eviction



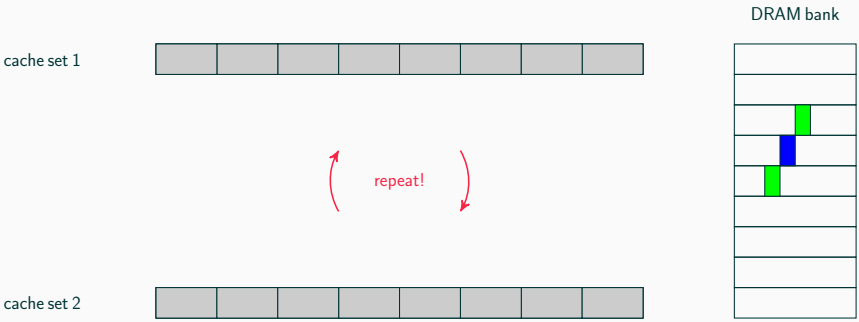
#2 Hammering with cache eviction



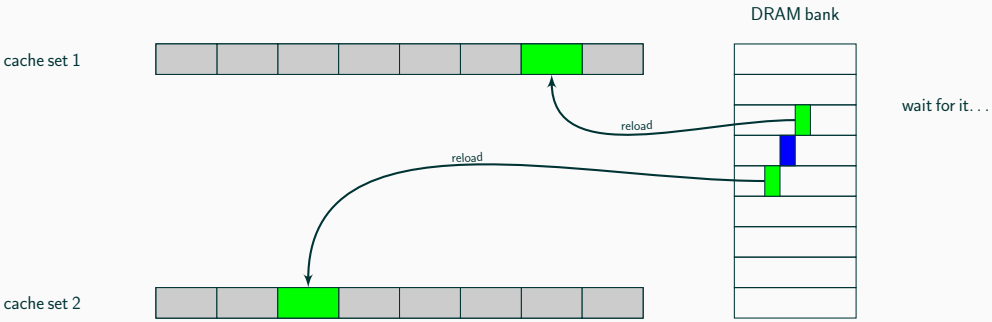
#2 Hammering with cache eviction



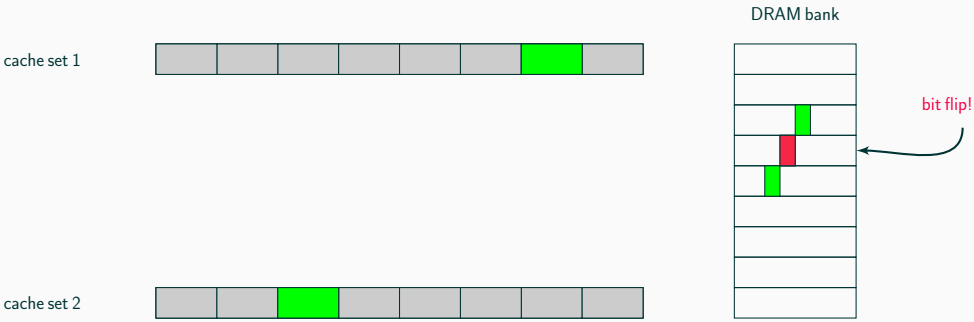
#2 Hammering with cache eviction



#2 Hammering with cache eviction



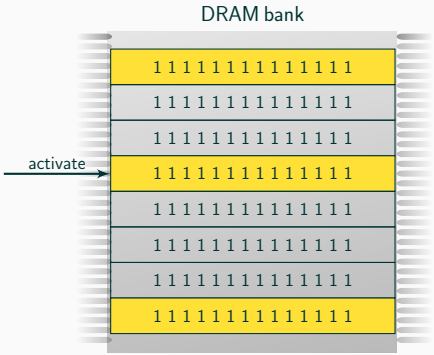
#2 Hammering with cache eviction



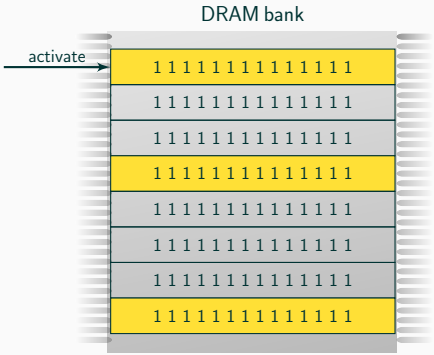
Hammering techniques

- There are three different hammering techniques
- #1: Hammer one row next to victim row and other random rows
- #2: Hammer two rows neighboring victim row
- #3: Hammer only one row next to victim row

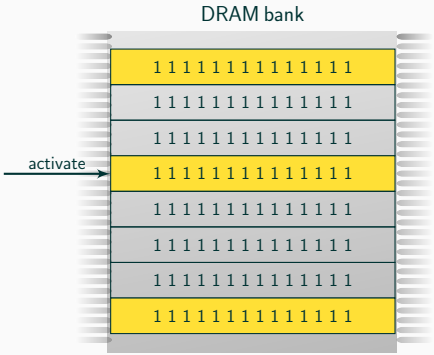
#1 - Single-sided hammering



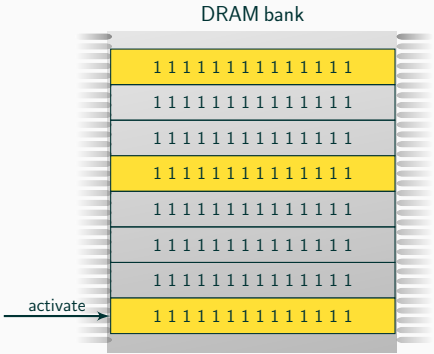
#1 - Single-sided hammering



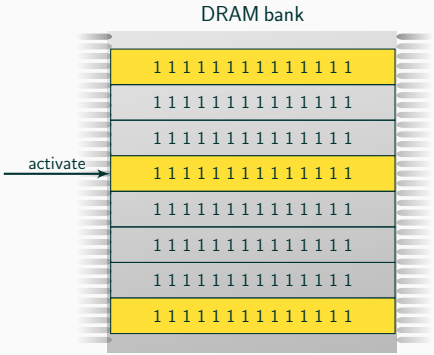
#1 - Single-sided hammering



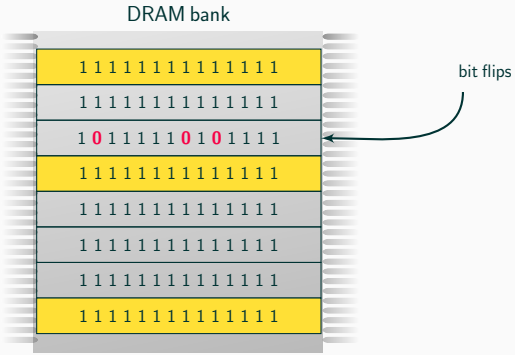
#1 - Single-sided hammering



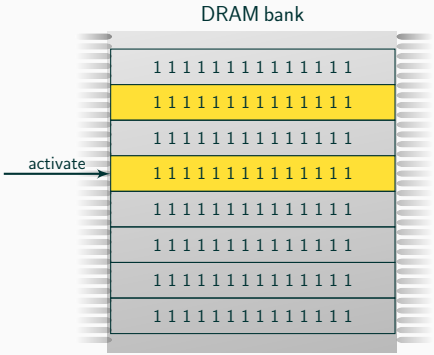
#1 - Single-sided hammering



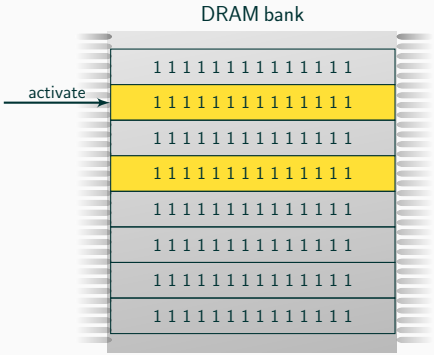
#1 - Single-sided hammering



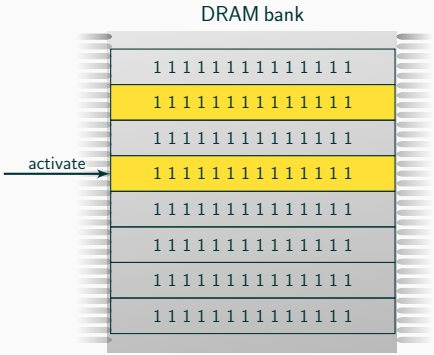
#2 - Double-sided hammering



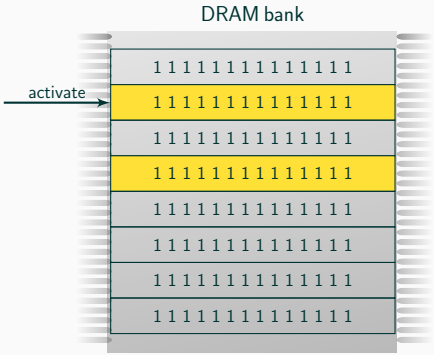
#2 - Double-sided hammering



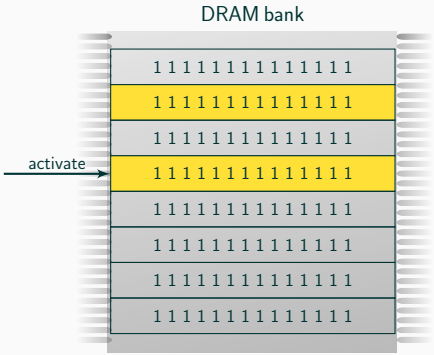
#2 - Double-sided hammering



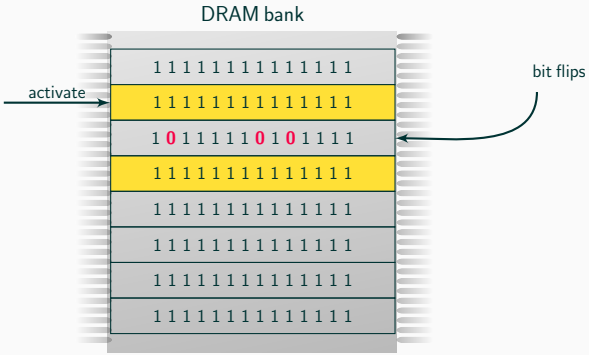
#2 - Double-sided hammering



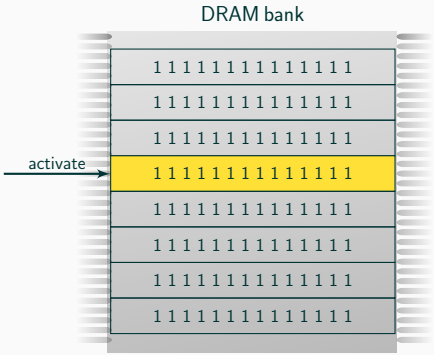
#2 - Double-sided hammering



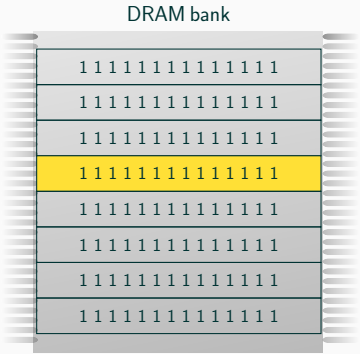
#2 - Double-sided hammering



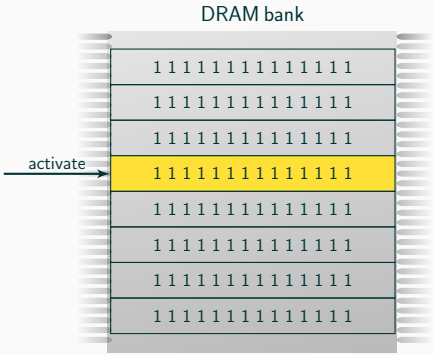
#3 - One-location hammering



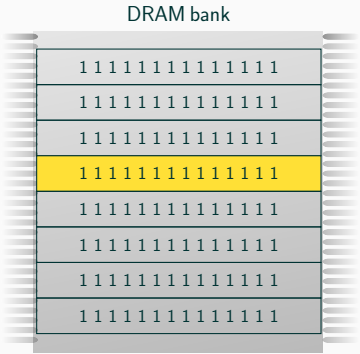
#3 - One-location hammering



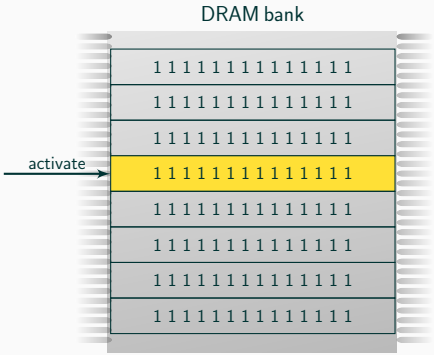
#3 - One-location hammering



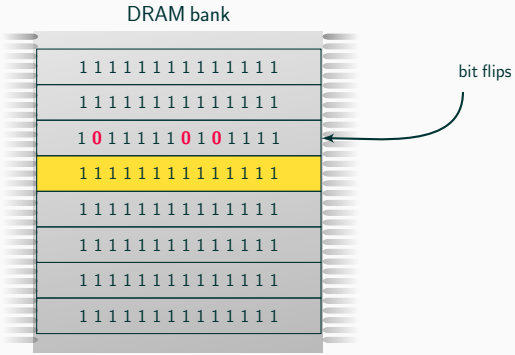
#3 - One-location hammering



#3 - One-location hammering



#3 - One-location hammering



One-location hammering

- We proposed one-location hammering

One-location hammering

- We proposed one-location hammering
- The only technique which accesses only one row

One-location hammering

- We proposed one-location hammering
- The only technique which accesses only one row
- Circumvents all countermeasures which expect accesses to multiple rows

One-location hammering

- We proposed one-location hammering
- The only technique which accesses only one row
- Circumvents all countermeasures which expect accesses to multiple rows
- Why does it work?

- **Open-page policy**: Keep row opened and buffered

- **Open-page policy**: Keep row opened and buffered
  - Beneficial for memory access latency, power consumption, bank utilization for a low number of memory accesses

- **Open-page policy**: Keep row opened and buffered
  - Beneficial for memory access latency, power consumption, bank utilization for a low number of memory accesses
- **Close-page policy**: Immediately close row, ready to open a new row

- **Open-page policy**: Keep row opened and buffered
  - Beneficial for memory access latency, power consumption, bank utilization for a low number of memory accesses
- **Close-page policy**: Immediately close row, ready to open a new row
  - Achieve better performance when a high number of memory accesses is performed

- **Open-page policy**: Keep row opened and buffered
  - Beneficial for memory access latency, power consumption, bank utilization for a low number of memory accesses
- **Close-page policy**: Immediately close row, ready to open a new row
  - Achieve better performance when a high number of memory accesses is performed
  - Perform better on multi-core systems David et al. 2011

- Policies that preemptively close rows, would allow one-location hammering

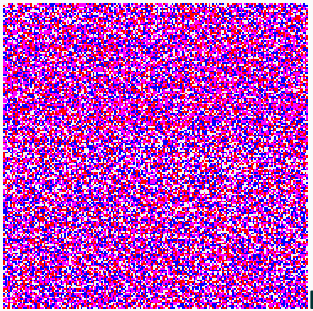
- Policies that preemptively close rows, would allow one-location hammering
- We observed close-page policies on desktop computers

- Policies that preemptively close rows, would allow one-location hammering
- We observed close-page policies on desktop computers
- Mobile devices (e.g., laptops) seem to use mostly open-page policies

How well does it work?

- Distribution of bit flips over 4kb-aligned memory regions
- Test each technique for 8 hours
- Scanned for bit flips after every hammering attempt
  - Hammering a random location of more than 100 000 randomly-chosen 4kB pages

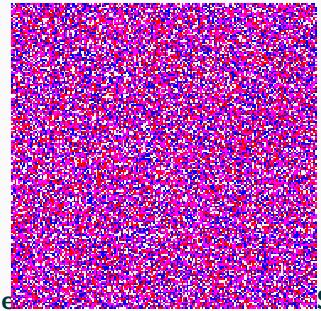
How well does it work?



Double-sided

77.0 % bit offsets

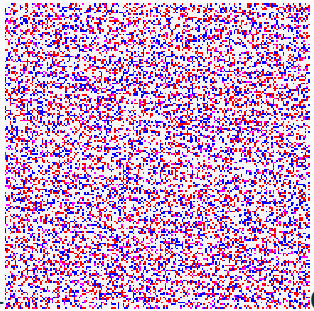
51.7 % 0→1 bit flips



Single-sided

78.5 % bit offsets

54.1 % 0→1 bit flips



One-location

36.5 % bit offsets

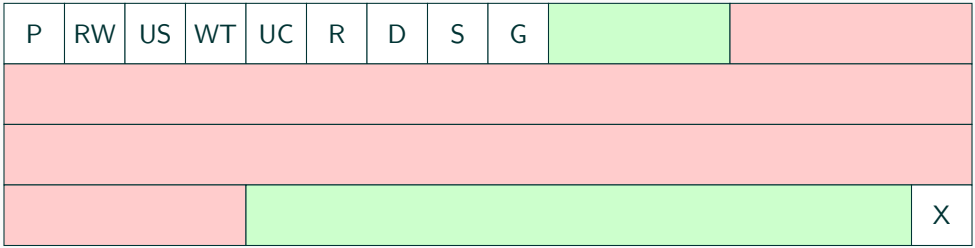
51.6 % 0→1 bit flips

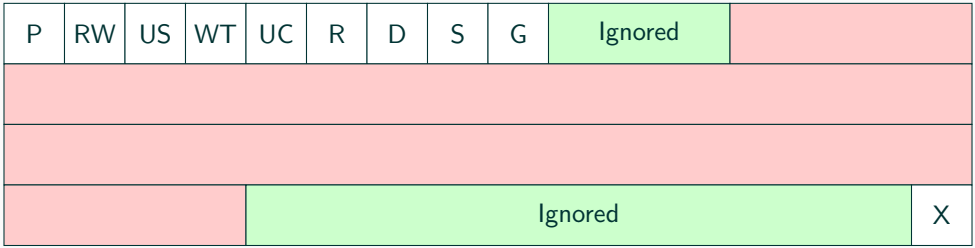
Exploits

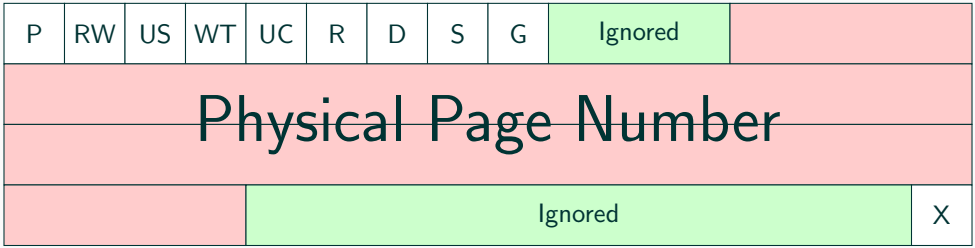
How to exploit random bit flips?

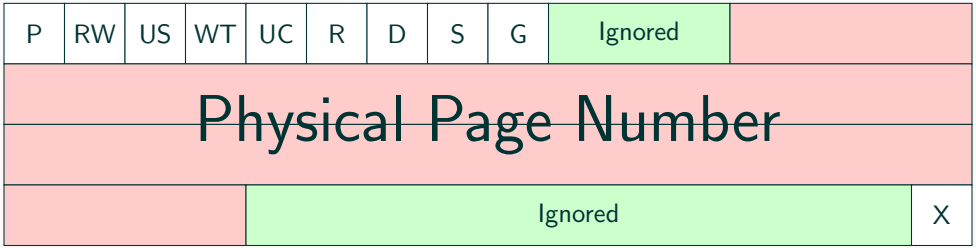
- They are not random $\rightarrow$ highly reproducible flip pattern!
  1. Choose a data structure that you can place at arbitrary memory locations
  2. Scan for “good” flips
  3. Place data structure there
  4. Trigger bit flip again

Page Table Entries




P
RW
US
WT
UC
R
D
S
G
Ignored
Ignored
X

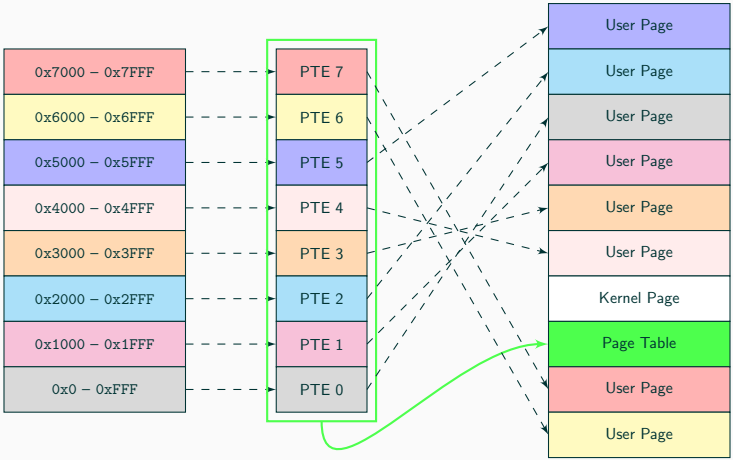

P
RW
US
WT
UC
R
D
S
G
Ignored
Physical Page Number
Ignored
X



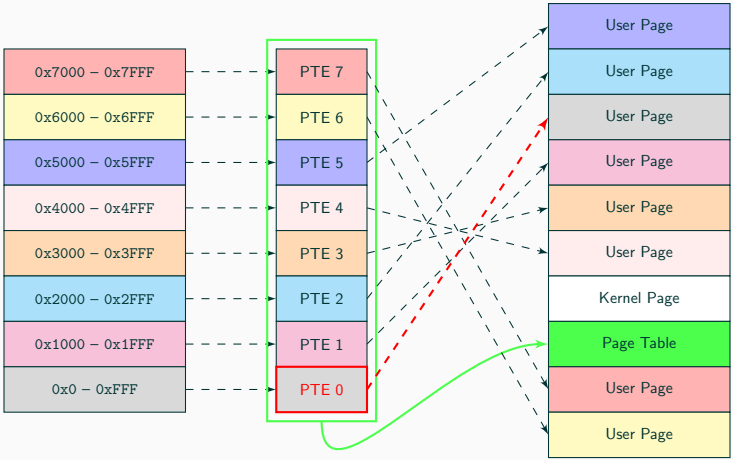
Each 4 KB page table consists of 512 such entries

Getting root by targeting the kernel

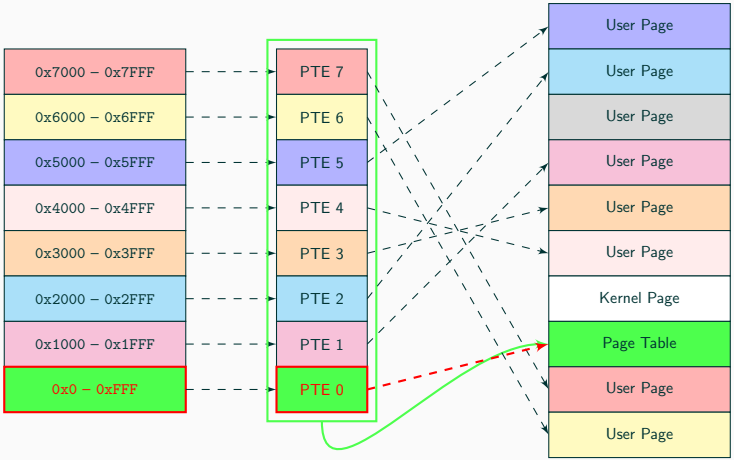
Page Table Manipulation



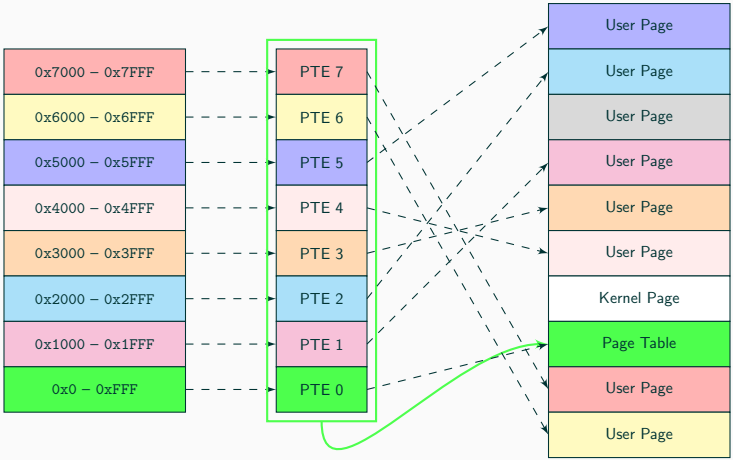
Page Table Manipulation



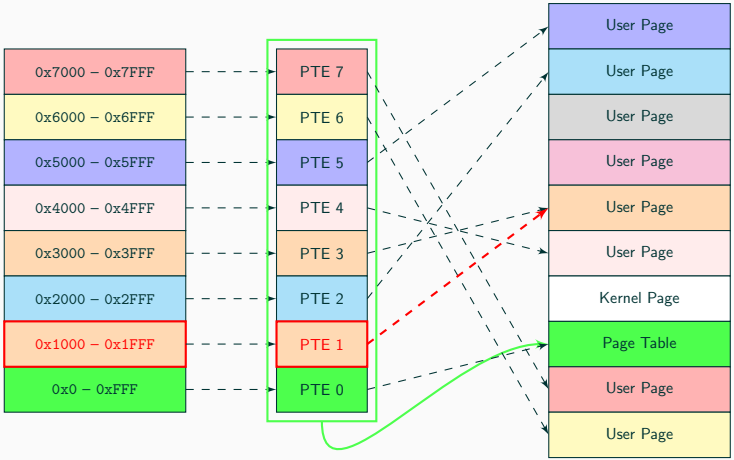
Page Table Manipulation



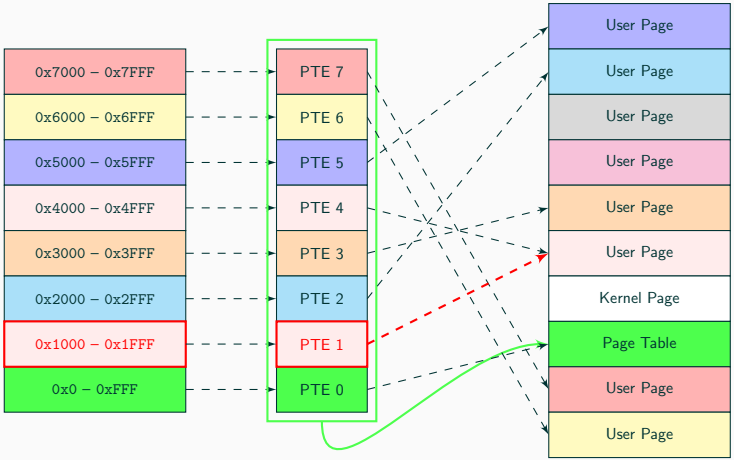
Page Table Manipulation



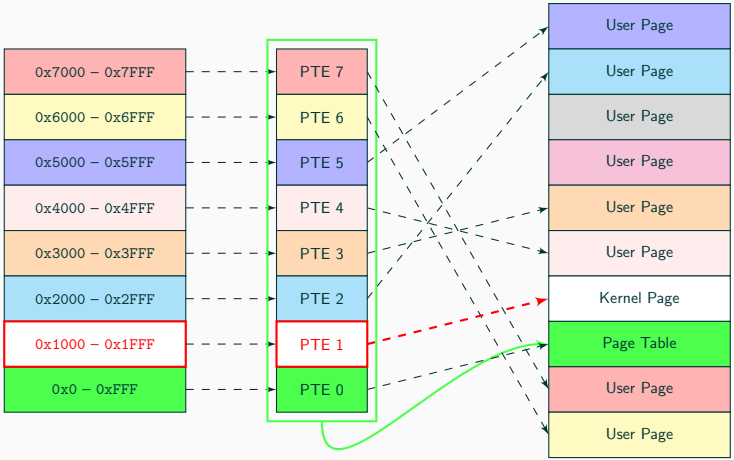
Page Table Manipulation



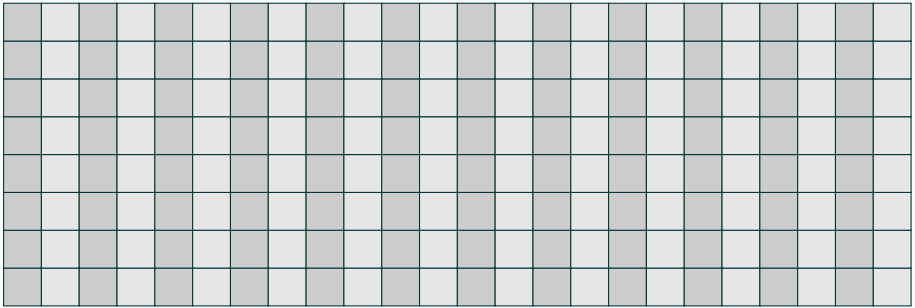
Page Table Manipulation



Page Table Manipulation

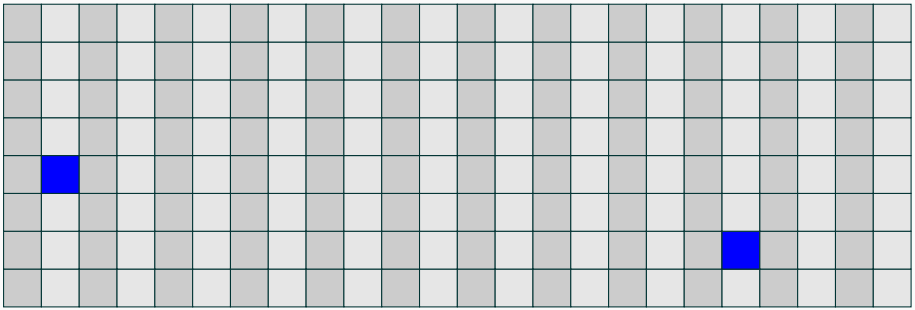


Search for page with flip

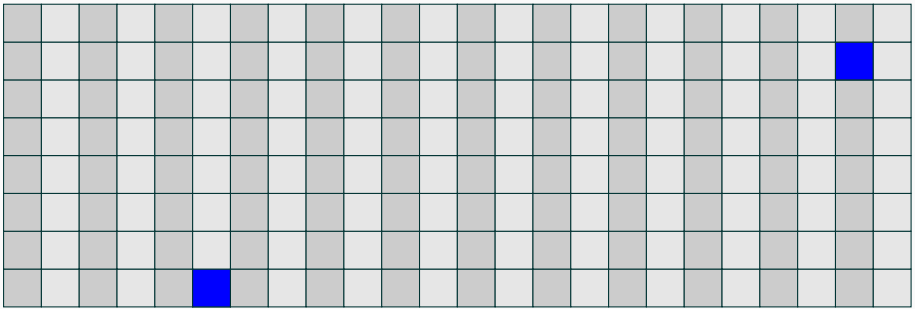


Hammering memory locations in different rows

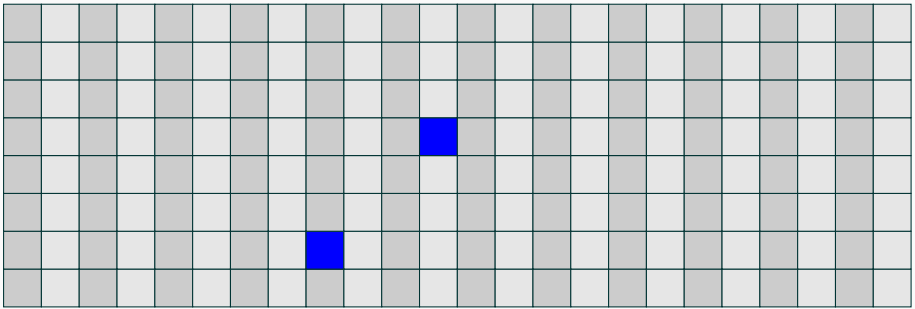
Search for page with flip



Hammering memory locations in different rows

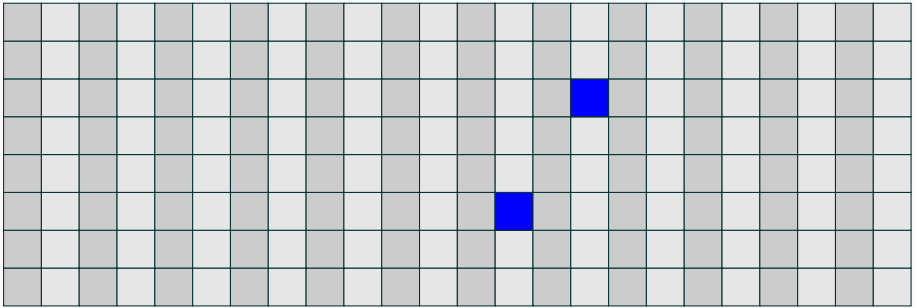


Hammering memory locations in different rows

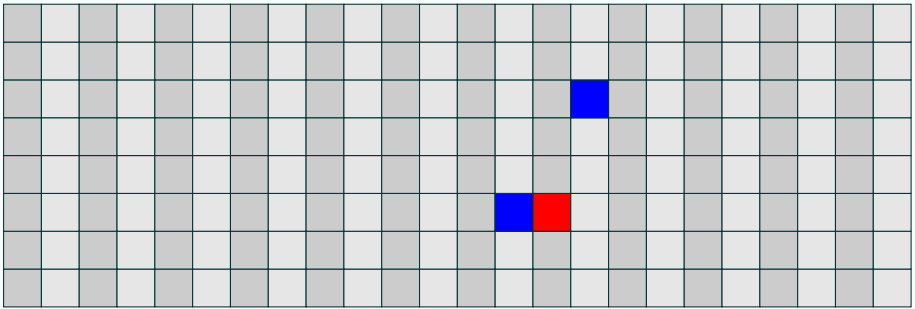


Hammering memory locations in different rows

Search for page with flip

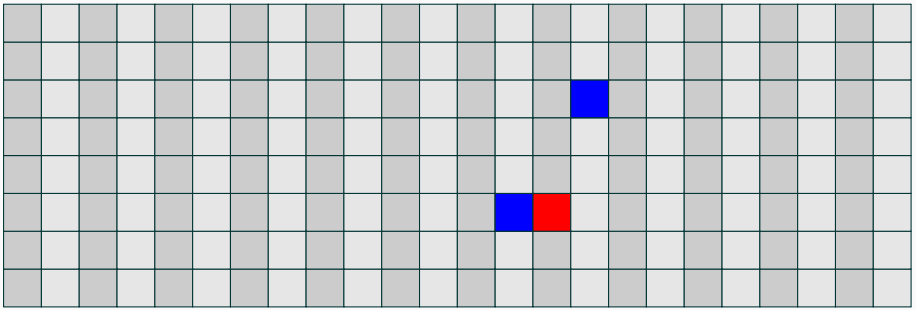


Hammering memory locations in different rows

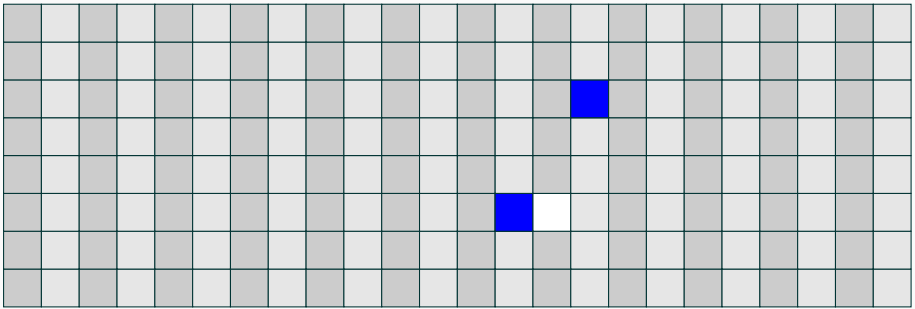


Hammering memory locations in different rows

Release page with flip

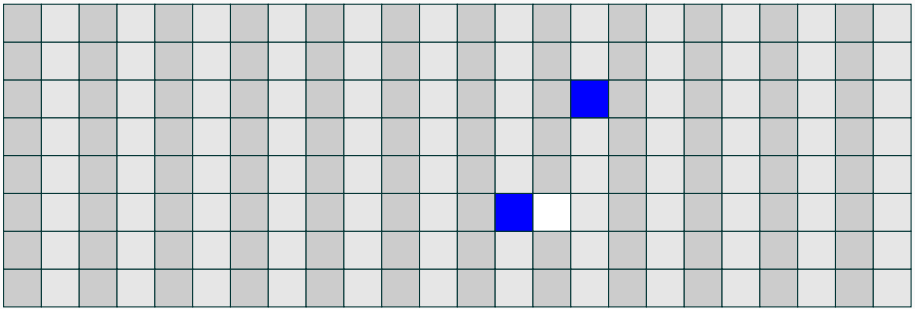


Release bit flip page



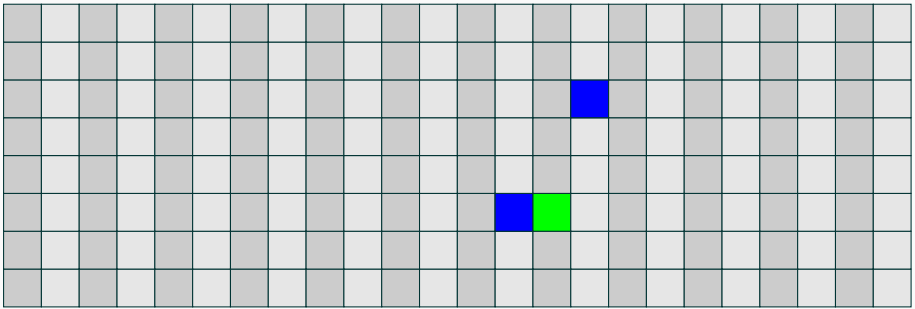
Release bit flip page

Fill all remaining memory with page tables



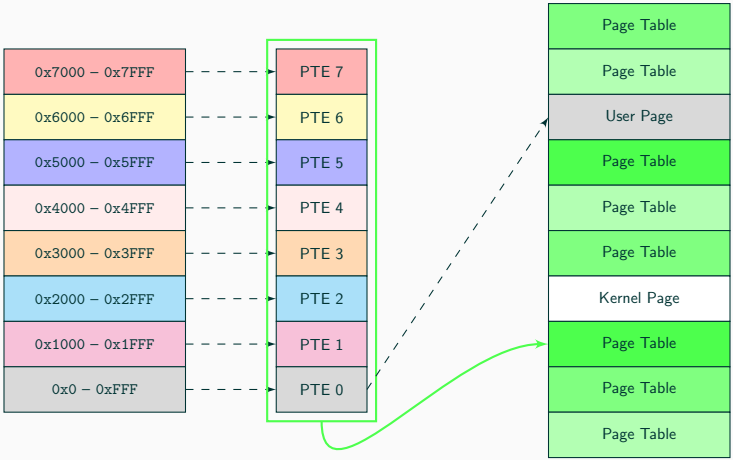
Place page table on bit flip page

Fill all remaining memory with page tables

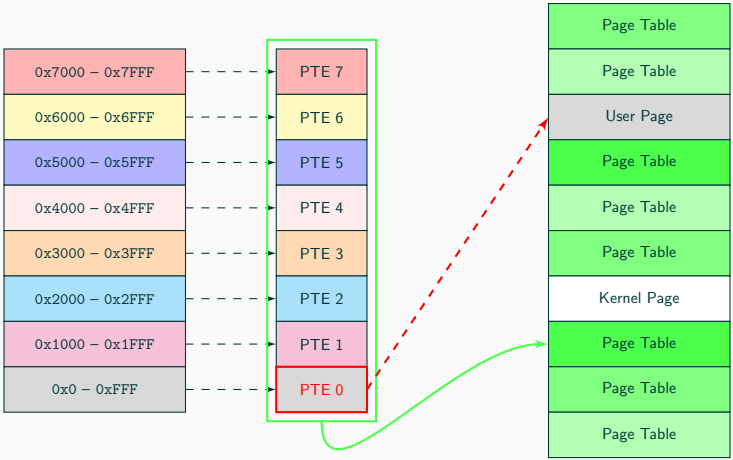


Place page table on bit flip page

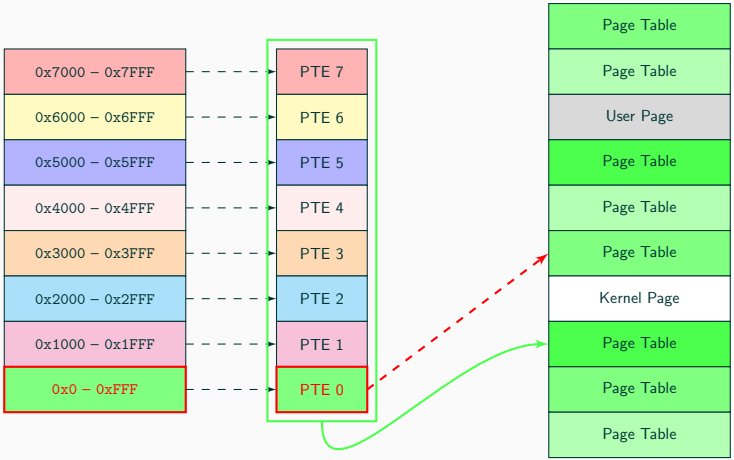
Page Table Manipulation



Page Table Manipulation



Page Table Manipulation



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 $\rightarrow$ kernel privileges

Flipping Page Table PPN bits

- Idea from Seaborn et al. 2015

Flipping Page Table PPN bits

- Idea from Seaborn et al. 2015
- Same idea applied in several other works:
  - Rowhammer.js (Gruss, Maurice, et al. 2016)
  - One bit flips, one cloud flops (Xiao et al. 2016)
  - Drammer (Veen et al. 2016)

What if we cannot target kernel pages?

Opcode Flipping

- Many applications perform actions as root

Opcode Flipping

- Many applications perform actions as root
- They can be used by unprivileged users as well

- Many applications perform actions as root
- They can be used by unprivileged users as well
- Implicitly: e.g., `ping` or `mount`

Opcode Flipping

- Many applications perform actions as root
- They can be used by unprivileged users as well
- Implicitly: e.g., `ping` or `mount`
- Explicitly: `sudo`

- Target `sudo`, as it is the easiest to exploit

Opcode Flipping

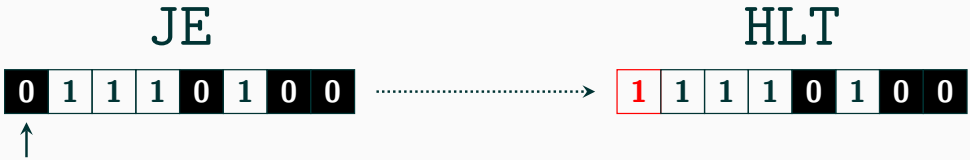
- Target `sudo`, as it is the easiest to exploit
- Use Rowhammer to circumvent password check

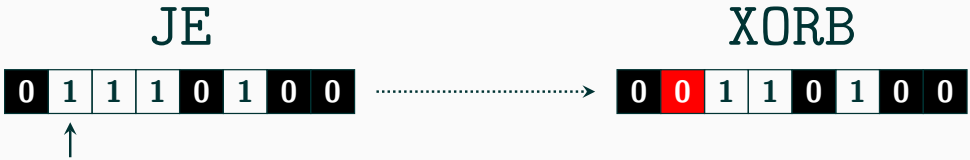
Opcode Flipping

- Target `sudo`, as it is the easiest to exploit
- Use Rowhammer to circumvent password check
- Changing program logic with bit flips

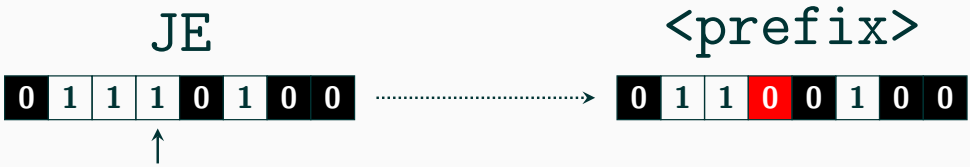
Opcode Flipping

- Target `sudo`, as it is the easiest to exploit
- Use Rowhammer to circumvent password check
- Changing program logic with bit flips
- What happens if we induce bit flips in instructions?

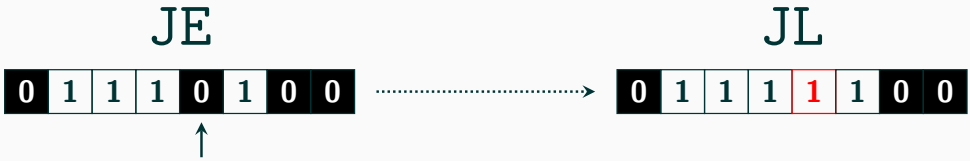

JE
0 1 1 1 0 1 0 0
HLT
1 1 1 1 0 1 0 0

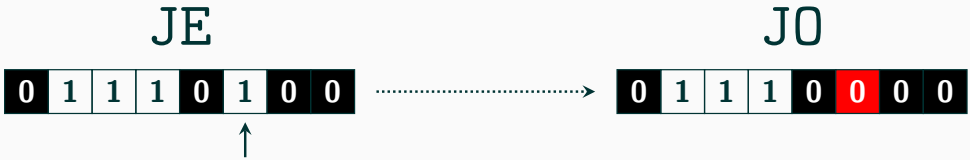

JE
0 1 1 1 0 1 0 0
XORB
0 0 1 1 0 1 0 0


JE
0 1 1 1 0 1 0 0
PUSHQ
0 1 0 1 0 1 0 0

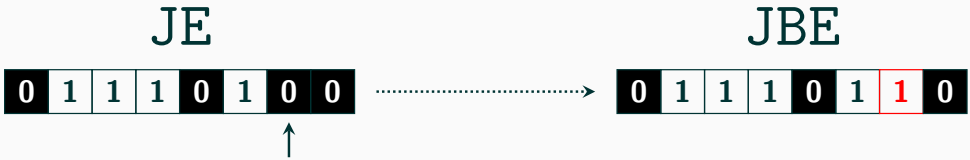

JE
0 1 1 1 0 1 0 0
<prefix>
0 1 1 0 0 1 0 0

Opcode Flipping - Conditional Jump

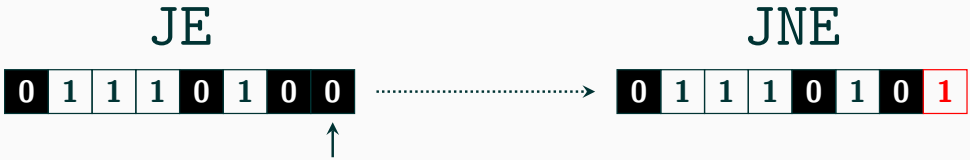



JE
0 1 1 1 0 1 0 0
JO
0 1 1 1 0 0 0 0

Opcode Flipping - Conditional Jump



Opcode Flipping - Conditional Jump



Opcode Flipping

- Conditional jumps are not the only targets

Opcode Flipping

- Conditional jumps are not the only targets
- Other targets include

Opcode Flipping

- Conditional jumps are not the only targets
- Other targets include
  - Comparisons
  - Addresses of memory loads/stores
  - Address calculations
  - ...

Opcode Flipping

- Conditional jumps are not the only targets
- Other targets include
  - Comparisons
  - Addresses of memory loads/stores
  - Address calculations
  - ...
- Manual analysis of `sudo` revealed 29 possible bitflips

Opcode Flipping

- Conditional jumps are not the only targets
- Other targets include
  - Comparisons
  - Addresses of memory loads/stores
  - Address calculations
  - ...
- Manual analysis of `sudo` revealed 29 possible bitflips
- They all somehow skipped the password check

- Remaining problem: maneuver the target binary page to a vulnerable physical page

Placing the binary

- Remaining problem: maneuver the target binary page to a vulnerable physical page
- Not as easy as with page tables

Placing the binary

- Remaining problem: maneuver the target binary page to a vulnerable physical page
- Not as easy as with page tables
- Many page tables can simply be sprayed over memory

Placing the binary

- Remaining problem: maneuver the target binary page to a vulnerable physical page
- Not as easy as with page tables
- Many page tables can simply be sprayed over memory
- There is only one copy of the binary in memory

- If a binary is loaded the first time, it is loaded to the memory

- If a binary is loaded the first time, it is loaded to the memory
- It stays in memory (in the page cache) even after execution

Page Cache

- If a binary is loaded the first time, it is loaded to the memory
- It stays in memory (in the page cache) even after execution
- Only evicted if page cache is full

Page Cache

- If a binary is loaded the first time, it is loaded to the memory
- It stays in memory (in the page cache) even after execution
- Only evicted if page cache is full
- Page cache is huge - usually all unused memory

Memory Waylaying

- We propose memory waylaying to move a victim page to a target page

- We propose memory waylaying to move a victim page to a target page
- By filling the page cache with executable pages, we evict the victim binary

Memory Waylaying

- We propose memory waylaying to move a victim page to a target page
- By filling the page cache with executable pages, we evict the victim binary
- Using the `mincore` function, we can check if the victim binary is evicted

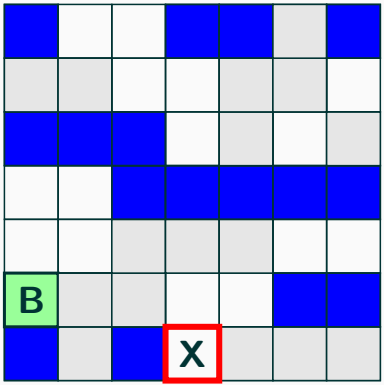
Memory Waylaying

- We propose memory waylaying to move a victim page to a target page
- By filling the page cache with executable pages, we evict the victim binary
- Using the `mincore` function, we can check if the victim binary is evicted
- Loading the victim binary results in a new physical page

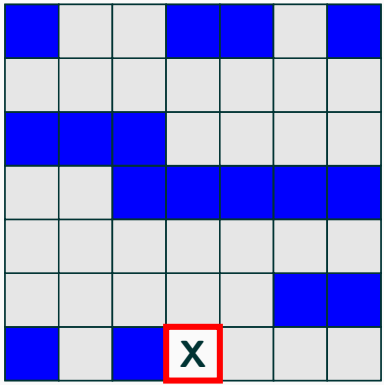
Memory Waylaying

- We propose memory waylaying to move a victim page to a target page
- By filling the page cache with executable pages, we evict the victim binary
- Using the `mincore` function, we can check if the victim binary is evicted
- Loading the victim binary results in a new physical page
- Continue until it is at the target page

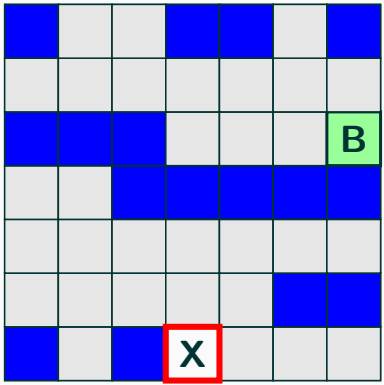
(1) Start



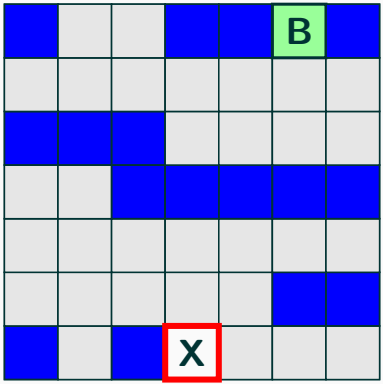
(2) Evict Page Cache



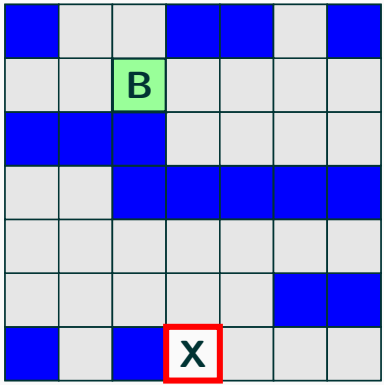
(3) Access Binary



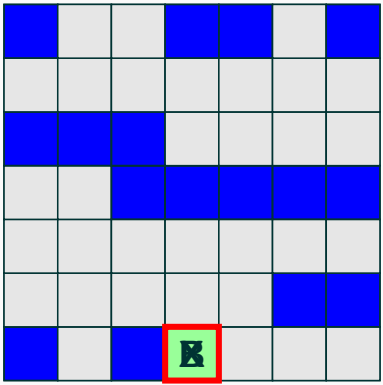
(4) Evict + Access



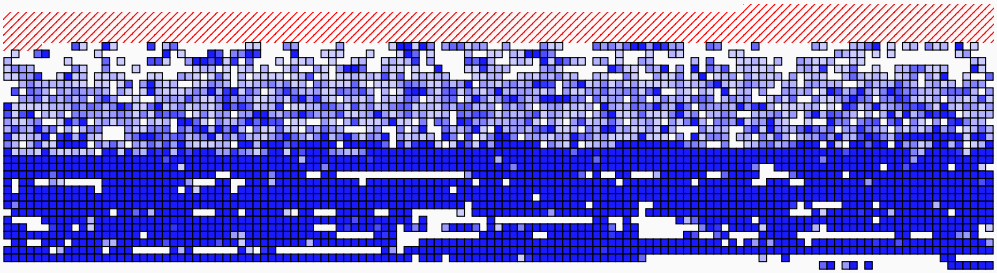
(5) Evict + Access



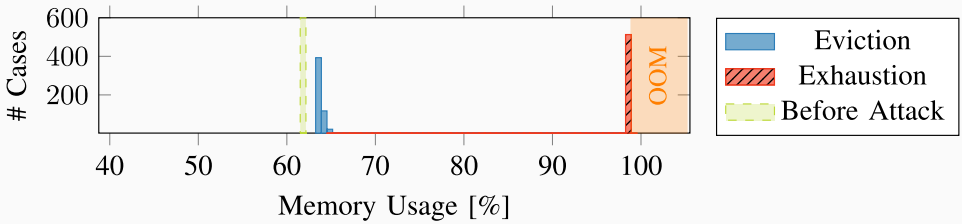
(6) Stop if target reached



- New pages cover most of the physical memory



- Great advantage over memory massaging: only negligible memory footprint



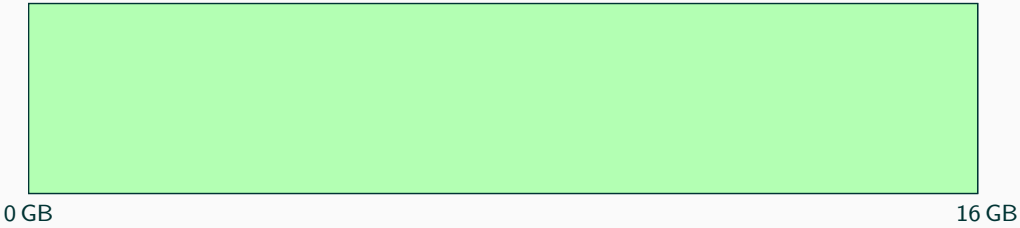
Other Techniques

- 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.

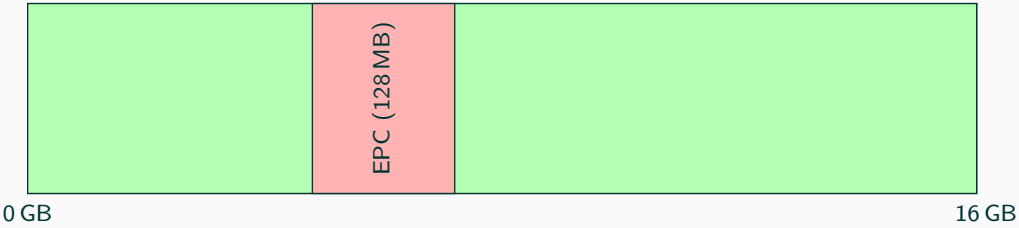
Rowhammer + SGX = Cheap Denial of Service

- SGX is an x86 instruction-set extension
- Integrity and Confidentiality of code and data in untrusted environments
- Run with user privileges and restricted, e.g., no system calls
- Execute programs in enclaves using protected areas of memory

SGX Encrypted Memory



SGX Encrypted Memory



- SGX explicitly protects against DRAM-based attacks
  - Cold-boot attacks
  - Memory bus snooping
  - Memory tampering attacks
- SGX EPC (Enclave Page Cache)
  - Region in DRAM
  - Cryptographically ensuring confidentiality, integrity and freshness of data
- Memory Encryption Engine (MEE)
  - Transparently encrypts memory

Bit Flips in the EPC

- What happens if a bit flips in the EPC?
- Integrity check will fail!

→ Lock's up the memory controller

→ Not a single further memory access!

→ System halts immediately

Bit Flips in the EPC

- What happens if a bit flips in the EPC?
- Integrity check will fail!

→ Lock's up the memory controller

→ Not a single further memory access!

→ System halts immediately

Sounds unsafe?

Bit Flips in the EPC

- What happens if a bit flips in the EPC?
- Integrity check will fail!

→ Lock's up the memory controller

→ Not a single further memory access!

→ System halts immediately

Sounds unsafe? It is unsafe!

Bit Flips in the EPC

- More and more services are deployed using SGX

- More and more services are deployed using SGX
- Cloud providers offer SGX support
  - Microsoft's Azure Confidential Computing

Bit Flips in the EPC

- More and more services are deployed using SGX
- Cloud providers offer SGX support
  - Microsoft's Azure Confidential Computing
- If a malicious enclave induces a bit flip, ...

Bit Flips in the EPC

- More and more services are deployed using SGX
- Cloud providers offer SGX support
  - Microsoft's Azure Confidential Computing
- If a malicious enclave induces a bit flip, ...
- ...the entire machine halts

Bit Flips in the EPC

- More and more services are deployed using SGX
- Cloud providers offer SGX support
  - Microsoft's Azure Confidential Computing
- If a malicious enclave induces a bit flip, ...
- ... the entire machine halts
- ... including co-located tenants

Bit Flips in the EPC

- More and more services are deployed using SGX
- Cloud providers offer SGX support
  - Microsoft's Azure Confidential Computing
- If a malicious enclave induces a bit flip, ...
- ...the entire machine halts
- ...including co-located tenants
- Denial-of-Service Attacks in the Cloud

SGX + One-location Hammering + Opcode Flipping = Undetectable Exploit

(Ab)using SGX Protection

- SGX protects software from malicious environments

(Ab)using SGX Protection

- SGX protects software from malicious environments
- Thwarts static and dynamic (= performance counters) analysis

(Ab)using SGX Protection

- SGX protects software from malicious environments
- Thwarts static and dynamic (= performance counters) analysis
- Hammering from SGX defeats countermeasures relying on this:
  - MASCAT
  - ANVIL
  - HexPADS
  - Herath and Fogh
  - Gruss et al.
  - Zhang et al.
  - Chiappetta et al.

One-location hammering

- Some countermeasures assume alternate access to at least two rows
- One-location hammering defeats countermeasures relying on this:
  - ANVIL
  - Corbet

Waylaying

- Some countermeasures assume that memory massaging requires a lot of memory
- Waylaying defeats countermeasures relying on this:
  - Gruss et al.
  - Van der Veen et al.

Opcode Flipping

- Some countermeasures assume kernel data structure have to be targeted
- Opcode flipping defeats countermeasures relying on this:
  - G-CATT

Opcode Flipping

Bypass \ Defense Class	Static Analysis	Performance Counters	Memory Access Pattern	Physical Proximity	Memory footprint
Intel SGX	●	●	○	○	○
One-location hammering	○	○	●	○	○
Opcode flipping	○	○	○	●	○
Memory waylaying	○	○	○	○	●
Defense class defeated	●	●	●	●	●

What do we learn from it?

- Many (academic) countermeasures were proposed to mitigate Rowhammer

What do we learn from it?

- Many (academic) countermeasures were proposed to mitigate Rowhammer
- We showed that all of them can be circumvented (Gruss, Lipp, et al. 2018)

What do we learn from it?

- Many (academic) countermeasures were proposed to mitigate Rowhammer
- We showed that all of them can be circumvented (Gruss, Lipp, et al. 2018)
- We cannot design countermeasures without completely understanding the attack

What do we learn from it?

- Many (academic) countermeasures were proposed to mitigate Rowhammer
- We showed that all of them can be circumvented (Gruss, Lipp, et al. 2018)
- We cannot design countermeasures without completely understanding the attack
- Otherwise we only patch concrete exploits, but do not solve the problem

What do we learn from it?

- We have to invest more into researching attacks

What do we learn from it?

- We have to invest more into researching attacks
- There are still aspects of Rowhammer we do not fully understand

What do we learn from it?

- We have to invest more into researching attacks
- There are still aspects of Rowhammer we do not fully understand
- However, this is required to design effective countermeasures

What do we learn from it?

- We have to invest more into researching attacks
- There are still aspects of Rowhammer we do not fully understand
- However, this is required to design effective countermeasures
- Moreover, new features might introduce new attack vectors (e.g., SGX)

What to we learn from it?

- We underestimated side-channel attacks for a long time

What to we learn from it?

- We underestimated side-channel attacks for a long time
- Industry and customers have to reconsider priorities $\rightarrow$ focus more on security instead of performance

What to we learn from it?

- We underestimated side-channel attacks for a long time
- Industry and customers have to reconsider priorities $\rightarrow$ focus more on security instead of performance
- Reliability issues (Rowhammer) can have security impacts

What to we learn from it?

- We underestimated side-channel attacks for a long time
- Industry and customers have to reconsider priorities $\rightarrow$ focus more on security instead of performance
- Reliability issues (Rowhammer) can have security impacts
- More research is required to understand attacks to ultimately mitigate them

Conclusion

- Rowhammer attacks have a huge attack potential
- One-location hammering showed us that previous assumptions on how to trigger the Rowhammer bug are invalid
  - Keeping only one DRAM row open is sufficient
- Op-code flipping (in the `sudo` binary) allows to obtain root privileges
- Proposed countermeasures in software are ineffective

Rowhammer: From the Basics to Sophisticated New Variants

Moritz Lipp, Michael Schwarz

February 20, 2018

Graz University of Technology