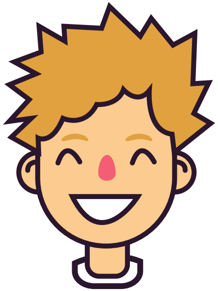
# Another Flip in the Row

Daniel Gruss, Moritz Lipp, Michael Schwarz

August 9, 2018

Graz University of Technology

## Who am I?



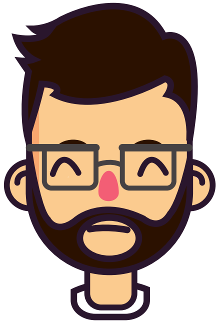
**Daniel Gruss**

PostDoc @ Graz University of Technology

@lavados

daniel.gruss@iaik.tugraz.at

## Who am I?



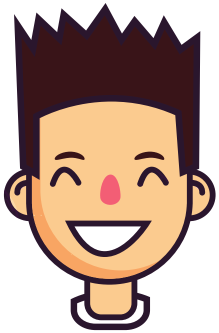
**Moritz Lipp**

PhD student @ Graz University of Technology

@mlxxyz

moritz.lipp@iaik.tugraz.at

## Who am I?



**Michael Schwarz**

PhD student @ Graz University of Technology

@misc0110

michael.schwarz@iaik.tugraz.at

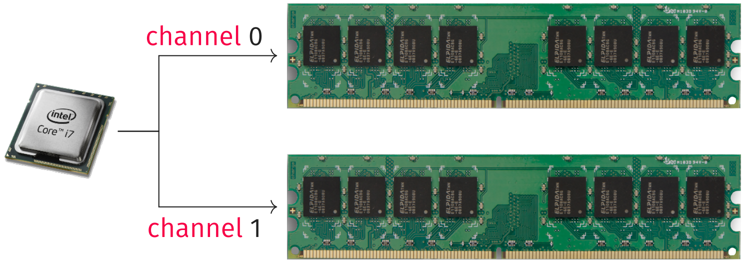
## And the rest of the team

The rest of the research team

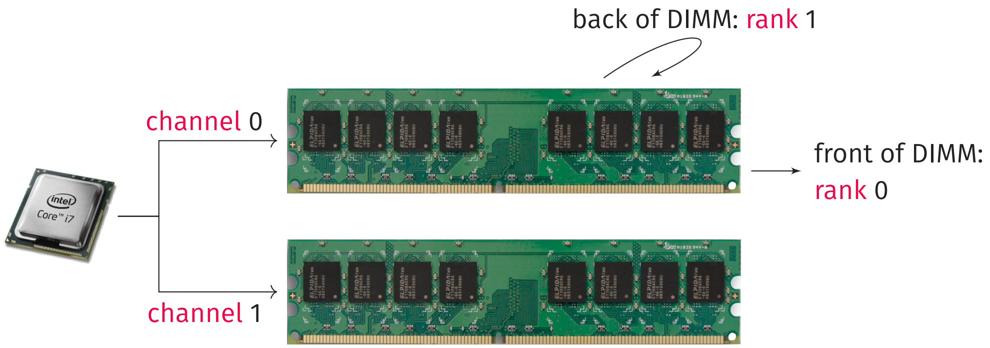
- Clémentine Maurice
- Daniel Genkin
- Jonas Juffinger
- Lukas Raab
- Lukas Lamster
- Misiker Tadesse Aga
- Sioli O’Connell
- Wolfgang Schoechl
- Yuval Yarom

## DRAM organization

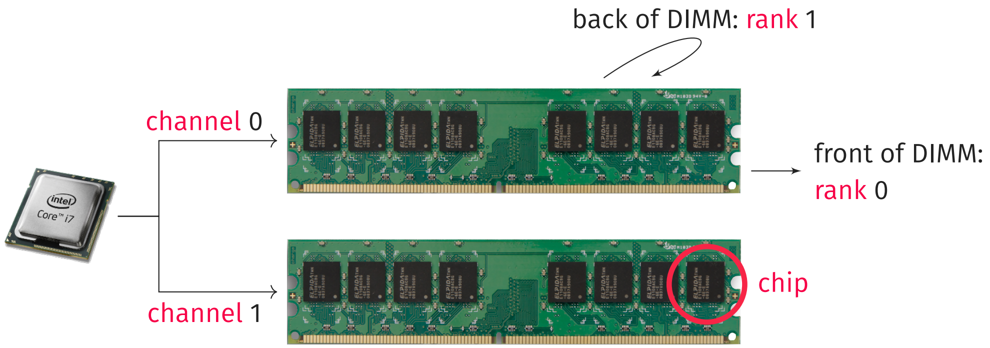
## DRAM organization



# DRAM organization

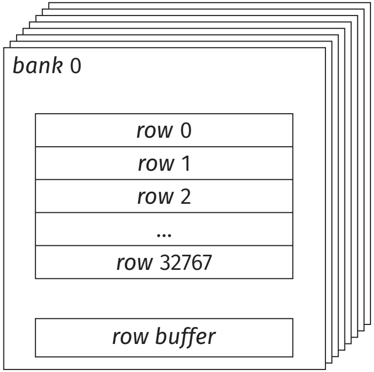


## DRAM organization

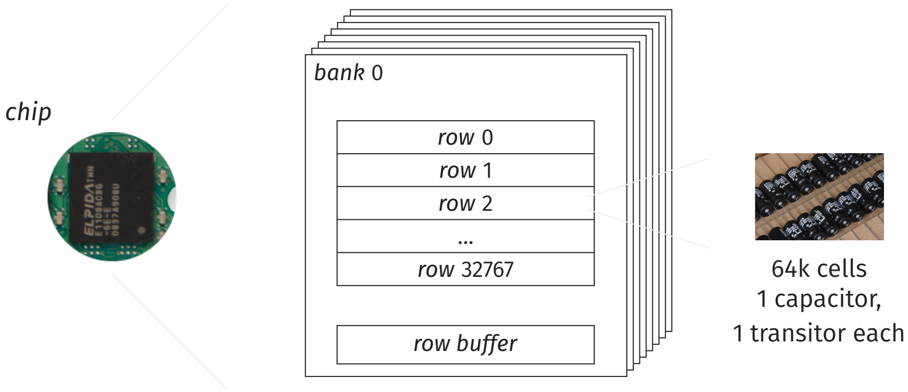


## DRAM organization

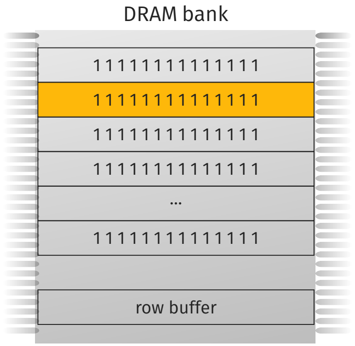
*chip*



# DRAM organization

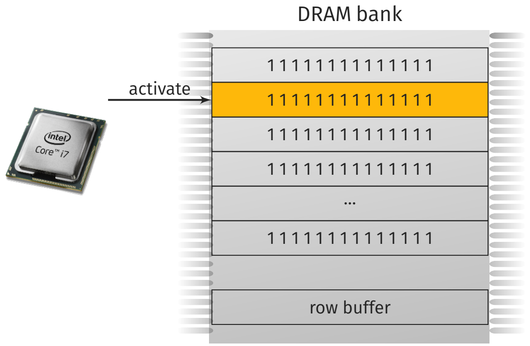


# How reading from DRAM works



CPU wants to access row 1

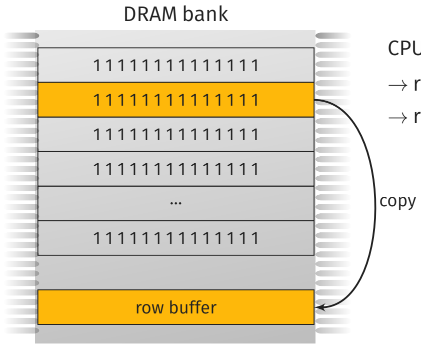
## How reading from DRAM works



CPU wants to access row 1

$\rightarrow$ row 1 activated

## How reading from DRAM works

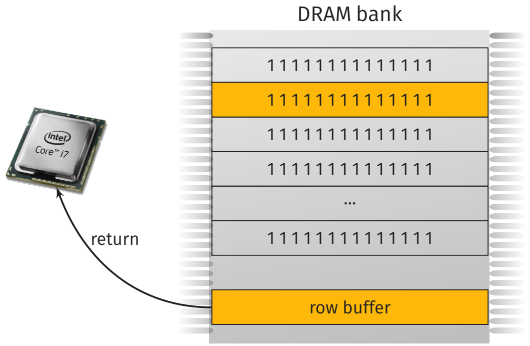


CPU wants to access row 1

→ row 1 activated

→ row 1 copied to row buffer

## How reading from DRAM works

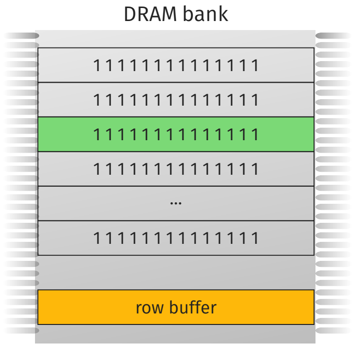


CPU wants to access row 1

→ row 1 activated

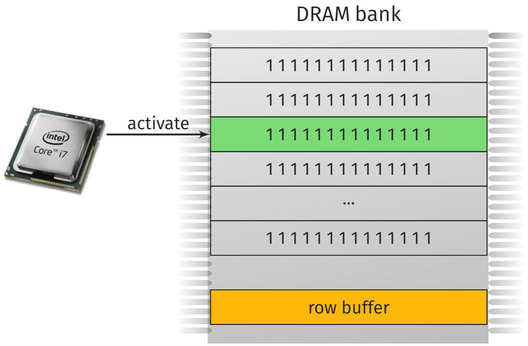
→ row 1 copied to row buffer

## How reading from DRAM works



CPU wants to access row 2

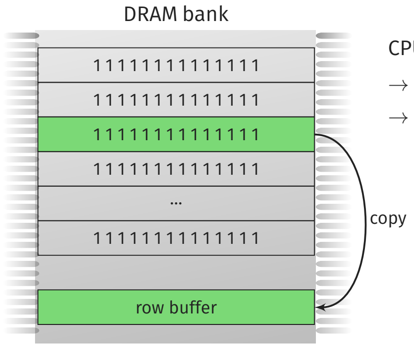
# How reading from DRAM works



CPU wants to access row 2

$\rightarrow$ row 2 activated

## How reading from DRAM works

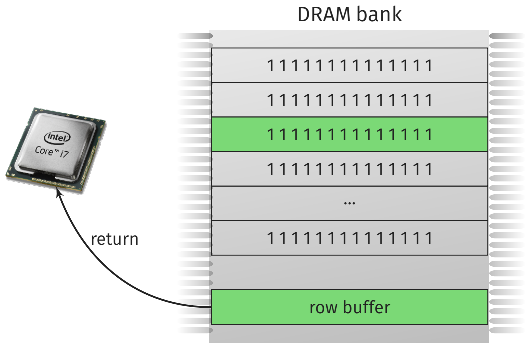


CPU wants to access row 2

$\rightarrow$ row 2 activated

$\rightarrow$ row 2 copied to row buffer

## How reading from DRAM works

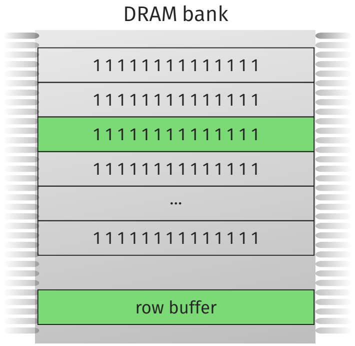


CPU wants to access row 2

$\rightarrow$ row 2 activated

$\rightarrow$ row 2 copied to row buffer

## How reading from DRAM works



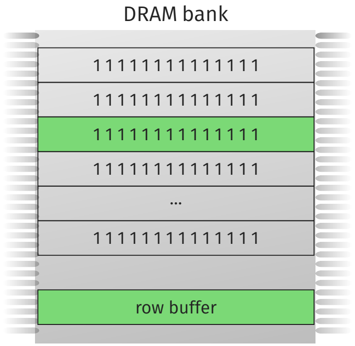
CPU wants to access row 2

→ row 2 activated

→ row 2 copied to row buffer

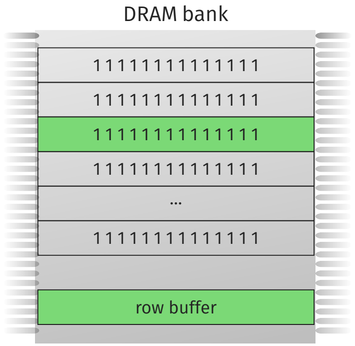
→ slow (row conflict)

## How reading from DRAM works



CPU wants to access row 2—again

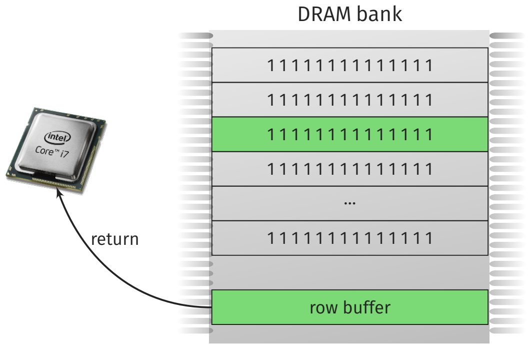
## How reading from DRAM works



CPU wants to access row 2—again

$\rightarrow$ row 2 already in row buffer

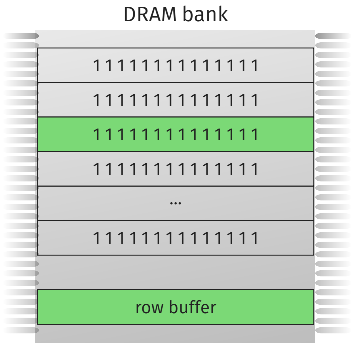
## How reading from DRAM works



CPU wants to access row 2—again

$\rightarrow$ row 2 already in row buffer

## How reading from DRAM works

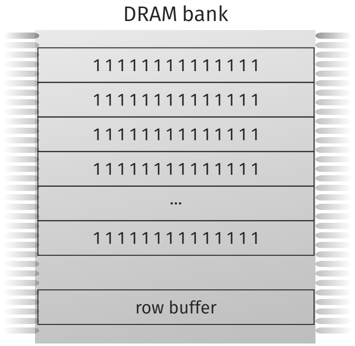


CPU wants to access row 2—again

→ row 2 already in row buffer

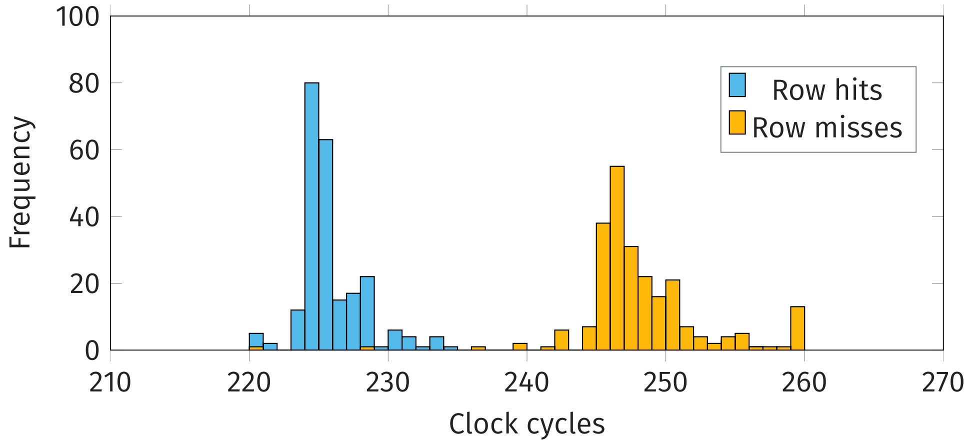
→ fast (row hit)

## How reading from DRAM works

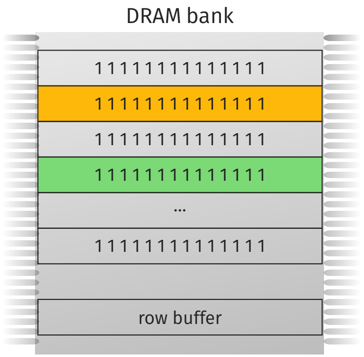


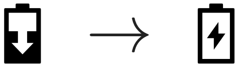
**row buffer = cache**

## Timing difference

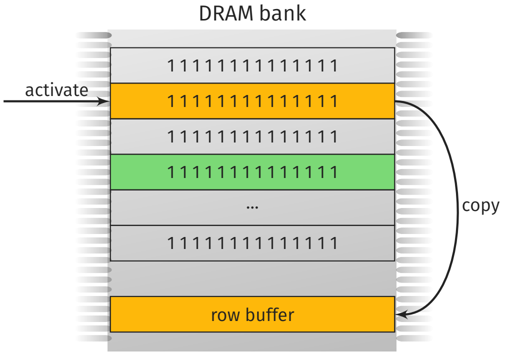


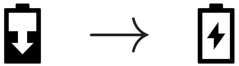
## Rowhammer


DRAM bank
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
...
1 1 1 1 1 1 1 1 1 1 1 1 1 1
row buffer

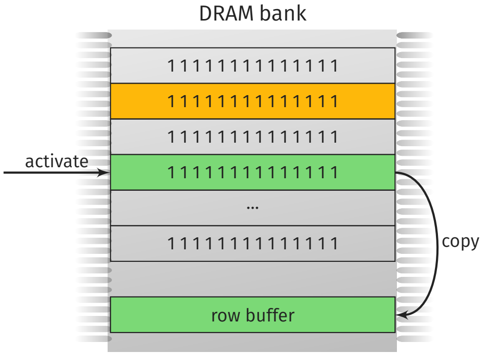


# Rowhammer


DRAM bank
activate
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
...
1 1 1 1 1 1 1 1 1 1 1 1 1 1
row buffer
copy

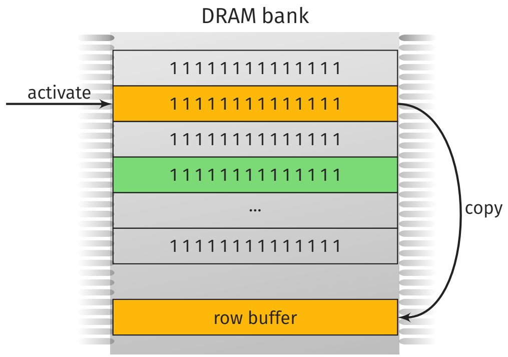


## Rowhammer



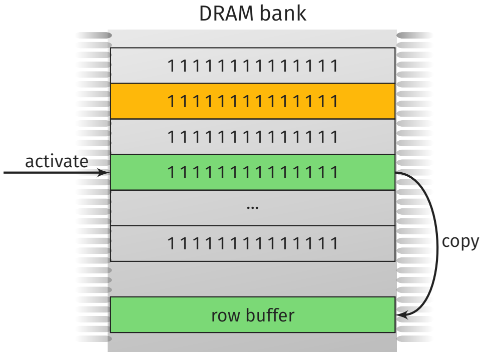
Cells leak faster upon proximate accesses → Rowhammer

# Rowhammer



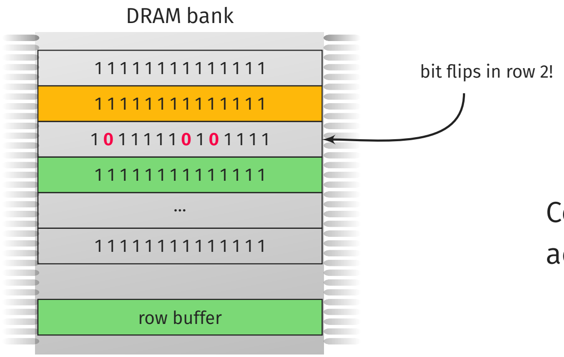
Cells leak faster upon proximate accesses $\rightarrow$ Rowhammer

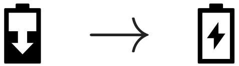
## Rowhammer



Cells leak faster upon proximate accesses → Rowhammer

## Rowhammer





Cells leak faster upon proximate accesses → Rowhammer

## How widespread is the issue?

## How widespread is the issue?

DDR3

DDR4

## How widespread is the issue?

DDR3

- 85% affected [Kim+14] (see Figure)

DDR4

## How widespread is the issue?

DDR3

- 85% affected [Kim+14] (see Figure)
- 52% affected [SD15]

DDR4

## How widespread is the issue?

DDR3

- 85% affected [Kim+14] (see Figure)
- 52% affected [SD15]

DDR4

- First believed to be safe

## How widespread is the issue?

DDR3

- 85% affected [Kim+14] (see Figure)
- 52% affected [SD15]

DDR4

- First believed to be safe
- We showed bit flips [Pes+16]

## How widespread is the issue?

DDR3

- 85% affected [Kim+14] (see Figure)
- 52% affected [SD15]

DDR4

- First believed to be safe
- We showed bit flips [Pes+16]
- 67% affected [Lan16]

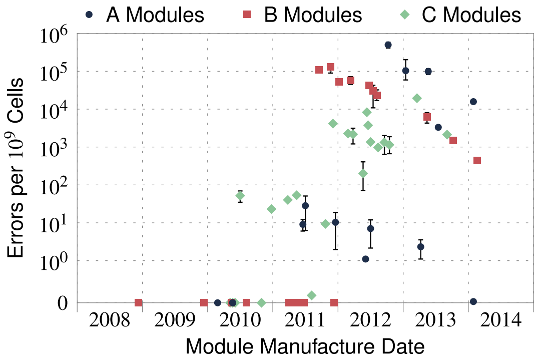
## How widespread is the issue?

DDR3

- 85% affected [Kim+14] (see Figure)
- 52% affected [SD15]

DDR4

- First believed to be safe
- We showed bit flips [Pes+16]
- 67% affected [Lan16]



BIT FLIPS
BIT FLIPS EVERYWHERE

## Requirements

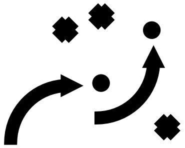
Memory accesses must be

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

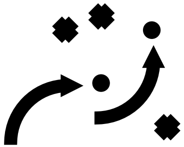
## How do we get enough uncached accesses?

## Access techniques

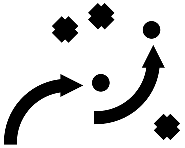
## Access techniques



## Access techniques

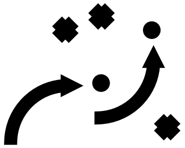


## Access techniques



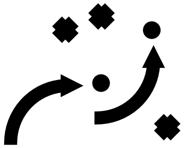
- `clflush` instruction → original paper [Kim+14]

## Access techniques



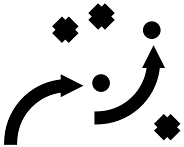
- `clflush` instruction → original paper [Kim+14]
- cache eviction [GMM16; Awe+16]

## Access techniques



- `clflush` instruction → original paper [Kim+14]
- cache eviction [GMM16; Awe+16]
- non-temporal accesses [QS16]

## Access techniques

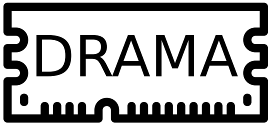


- `clflush` instruction $\rightarrow$ original paper [Kim+14]
- cache eviction [GMM16; Awe+16]
- non-temporal accesses [QS16]
- uncached memory [Vee+16]

## How do we target accesses?

## Physical addresses and DRAM

## Physical addresses and DRAM



## Physical addresses and DRAM

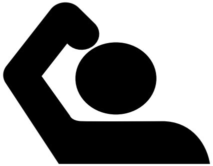
**DRAMA: How your DRAM becomes a security problem**

Anders Fogh & Michael Schwarz

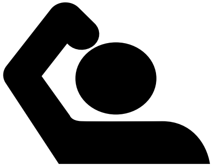
Black Hat Europe 2016

## How to exploit random bit flips?

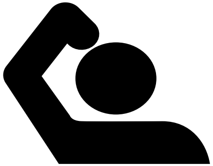
## How to exploit random bit flips?



## How to exploit random bit flips?

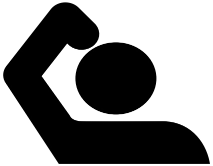


## How to exploit random bit flips?



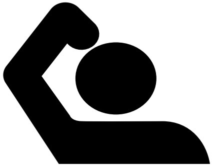
- They are not random $\rightarrow$ highly reproducible flip pattern!

## How to exploit random bit flips?



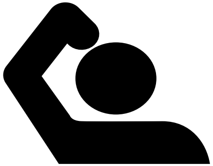
- They are not random $\rightarrow$ highly reproducible flip pattern!
    1. Choose a data structure that you can place at arbitrary memory locations

## How to exploit random bit flips?



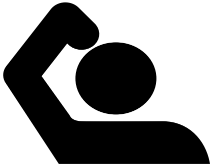
- They are not random $\rightarrow$ highly reproducible flip pattern!
  1. Choose a data structure that you can place at arbitrary memory locations
  2. Scan for "good" flips

## How to exploit random bit flips?



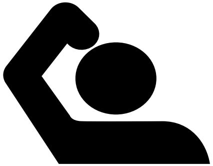
- They are not random $\rightarrow$ highly reproducible flip pattern!
  1. Choose a data structure that you can place at arbitrary memory locations
  2. Scan for “good” flips
  3. Place data structure there

## 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

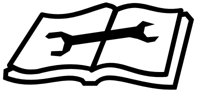
## How to exploit random bit flips?



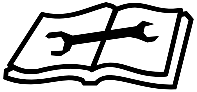
- They are not random $\rightarrow$ highly reproducible flip pattern!
  1. Choose a data structure that you can place at arbitrary memory locations
  2. Scan for “good” flips
  3. Place data structure there
  4. Trigger bit flip again
- Alternatively: Build a PUF [Ana+18]

## Strategy: Modify instructions

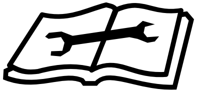
## Strategy: Modify instructions



## Strategy: Modify instructions

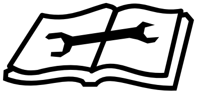


## Strategy: Modify instructions



- Idea from [SD15]

## Strategy: Modify instructions

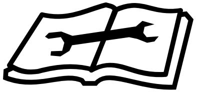


- Idea from [SD15]
- x86 op codes are variable length

## Strategy: Modify instructions

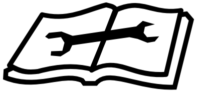
- Idea from [SD15]
- x86 op codes are variable length
    - Unsafe op codes (syscall) $\in$ safe but long multi-byte op codes

## Strategy: Modify instructions



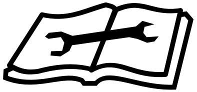
- Idea from [SD15]
- x86 op codes are variable length
  - Unsafe op codes (syscall) $\in$ safe but long multi-byte op codes
  - Only a problem with jumps to arbitrary addresses

## Strategy: Modify instructions



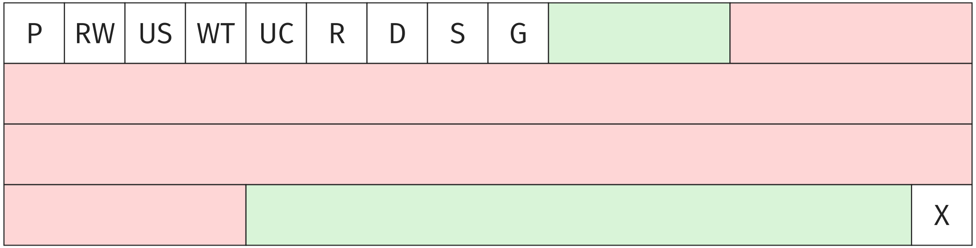
- Idea from [SD15]
- x86 op codes are variable length
    - Unsafe op codes (syscall) $\in$ safe but long multi-byte op codes
    - Only a problem with jumps to arbitrary addresses
- Flip a bit in a validated NaCl instruction sequence

## Strategy: Modify instructions

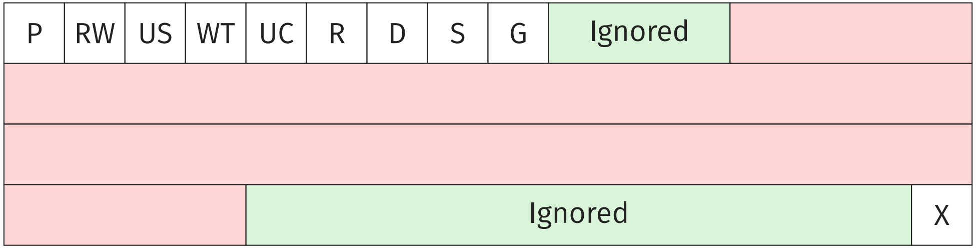


- Idea from [SD15]
- x86 op codes are variable length
  - Unsafe op codes (syscall) $\in$ safe but long multi-byte op codes
  - Only a problem with jumps to arbitrary addresses
- Flip a bit in a validated NaCl instruction sequence
  - Safe + validated jump $\rightarrow$ arbitrary jump

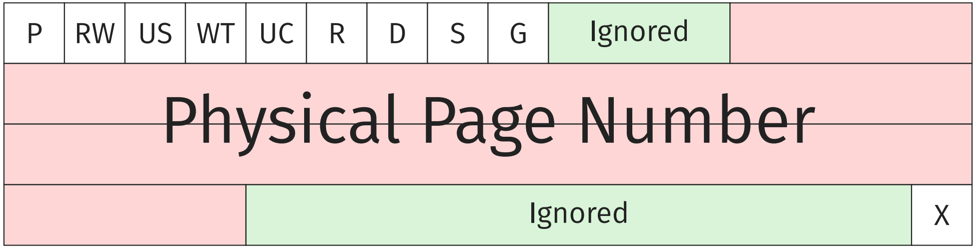
## Page Table Entries



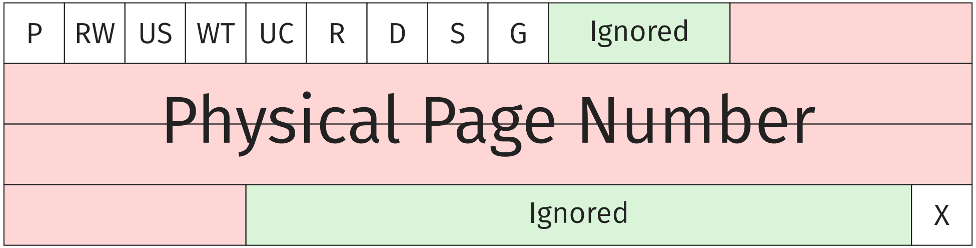
## Page Table Entries



## Page Table Entries

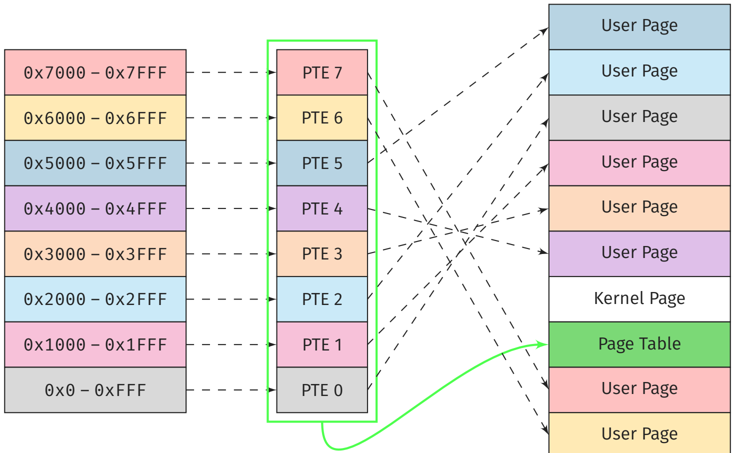


## Page Table Entries

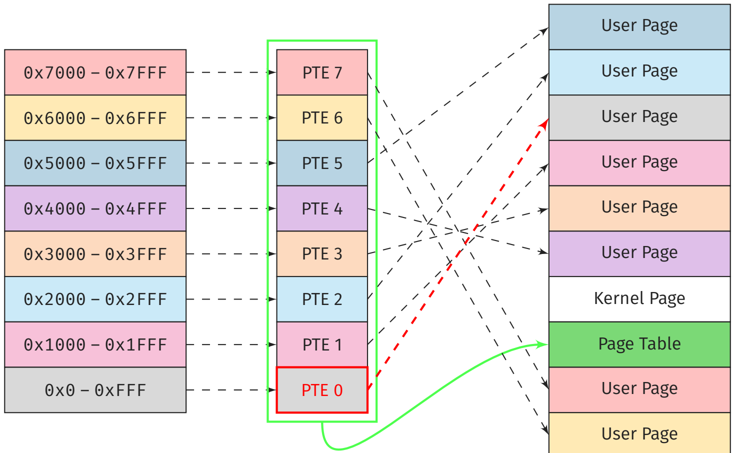


Each 4 KB page table consists of 512 such entries

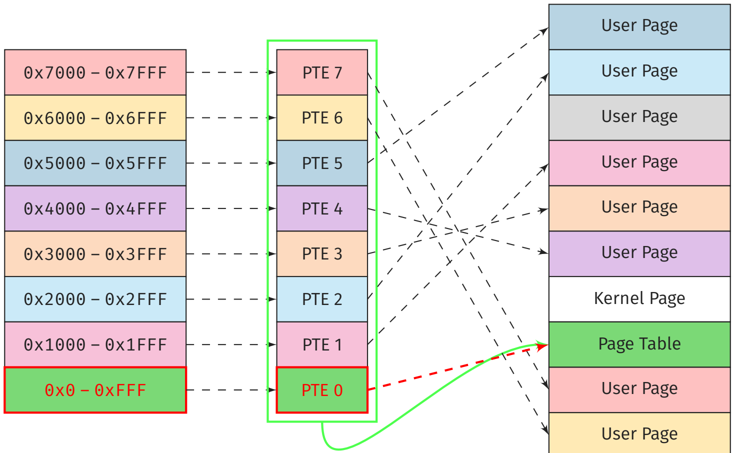
## Page Table Manipulation



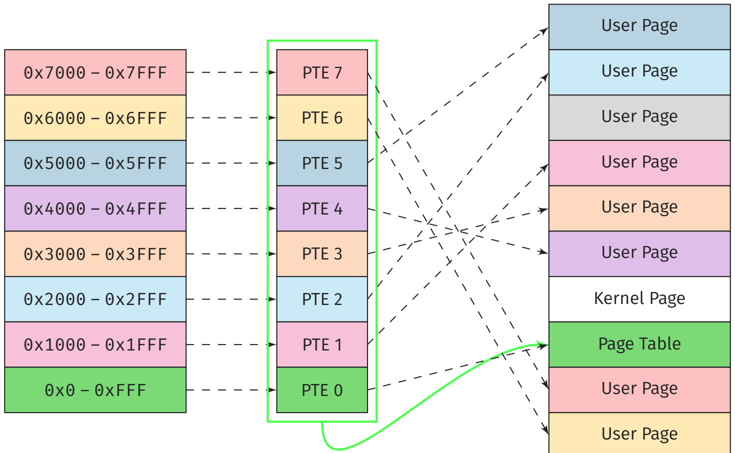
## Page Table Manipulation



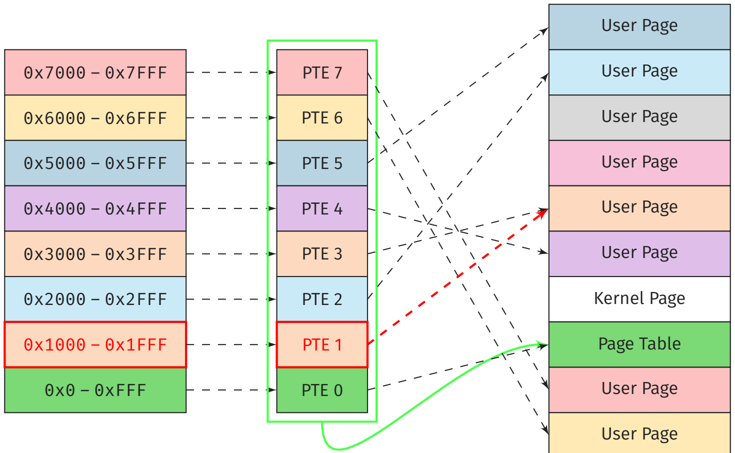
# Page Table Manipulation



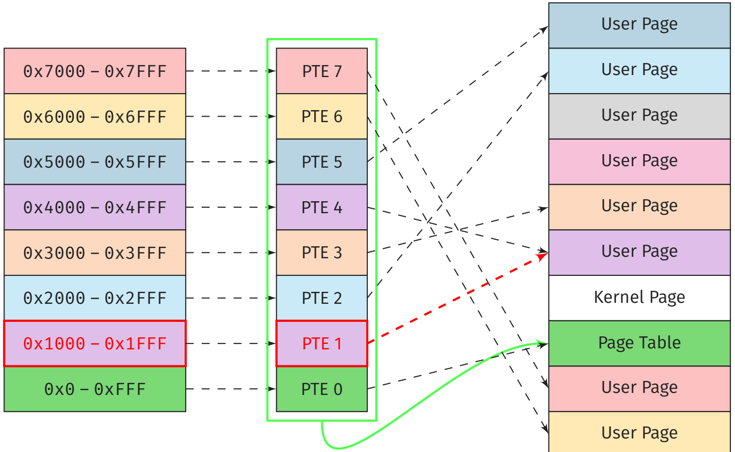
# Page Table Manipulation



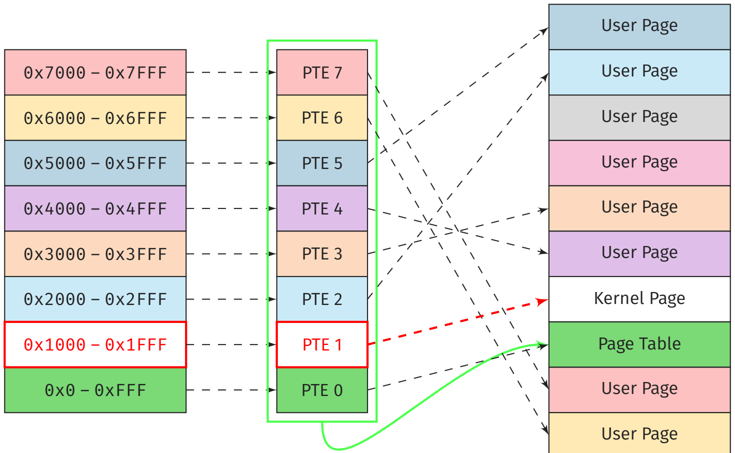
## Page Table Manipulation



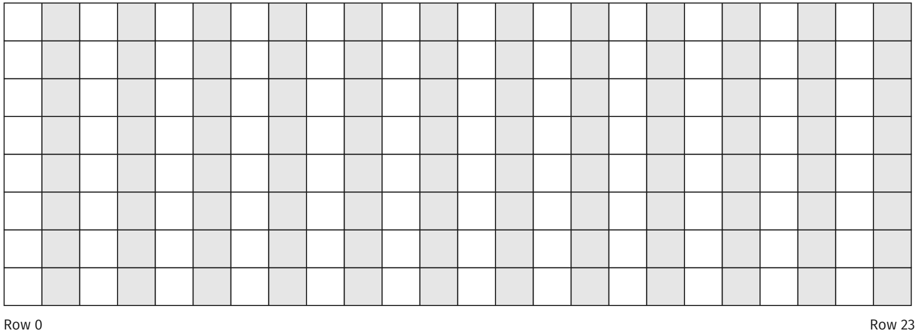
## Page Table Manipulation



# Page Table Manipulation

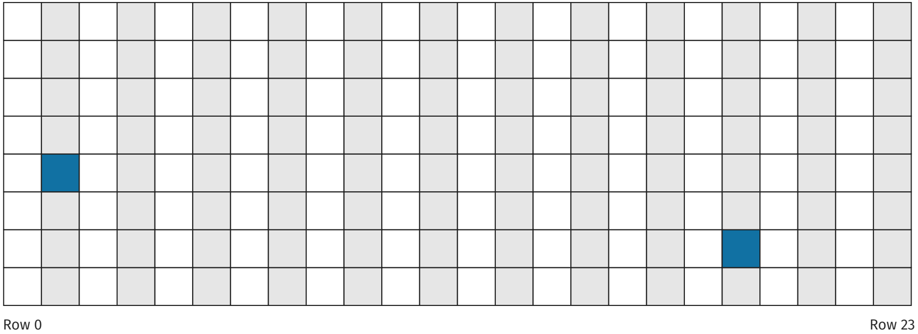


## Search for page with flip



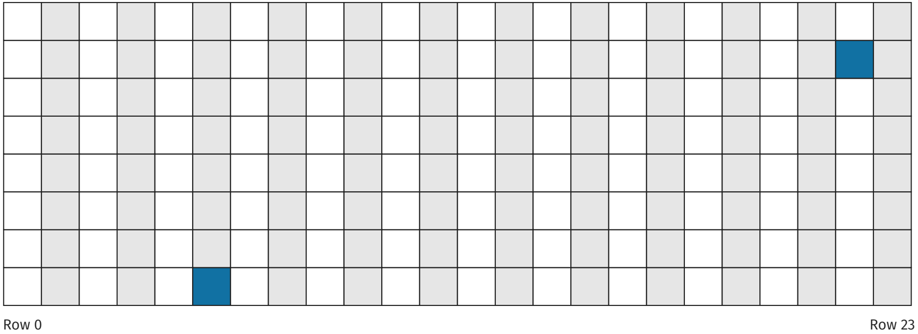
Hammering memory locations in different rows

## Search for page with flip



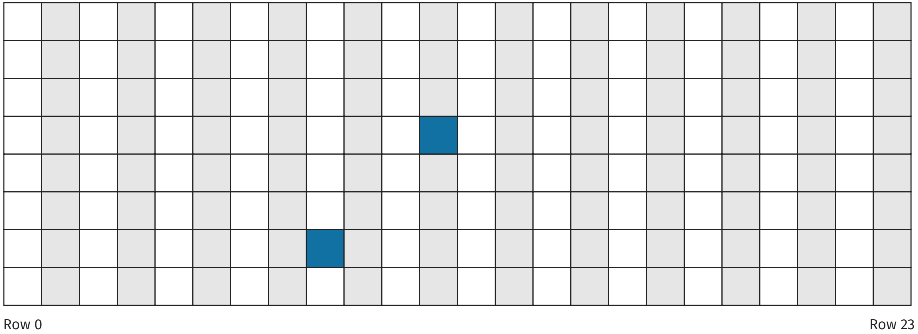
Hammering memory locations in different rows

## Search for page with flip



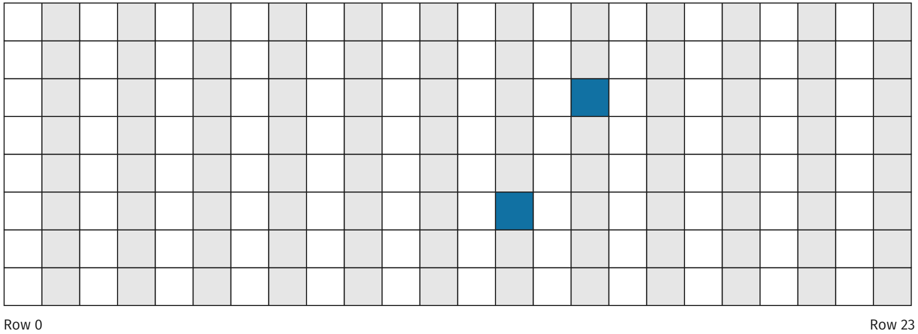
Hammering memory locations in different rows

## Search for page with flip



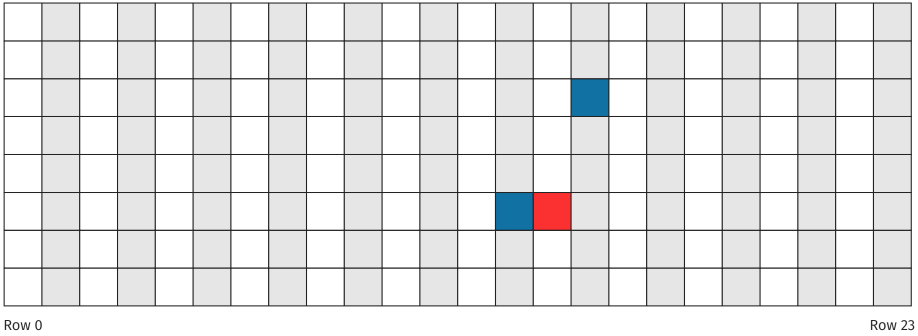
Hammering memory locations in different rows

# Search for page with flip



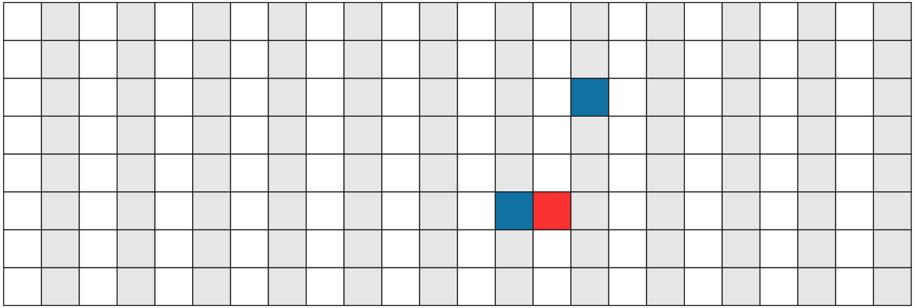
Hammering memory locations in different rows

## Search for page with flip

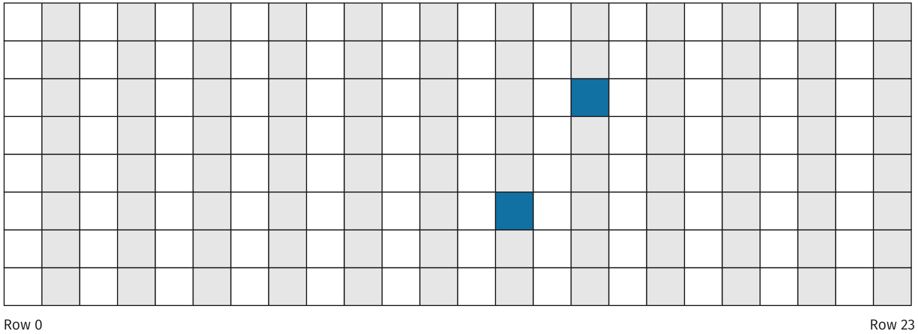


Hammering memory locations in different rows

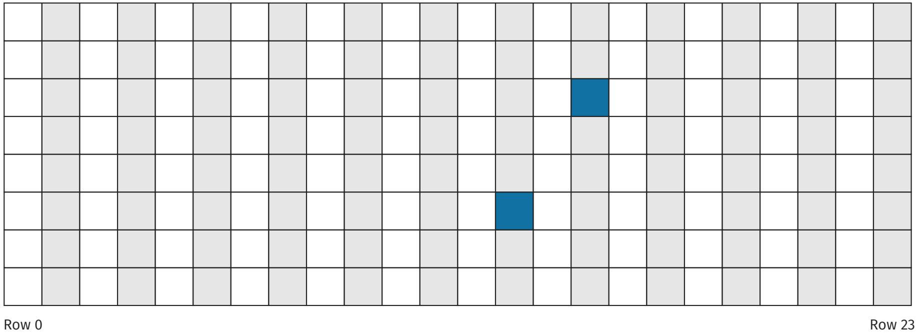
## Release page with flip



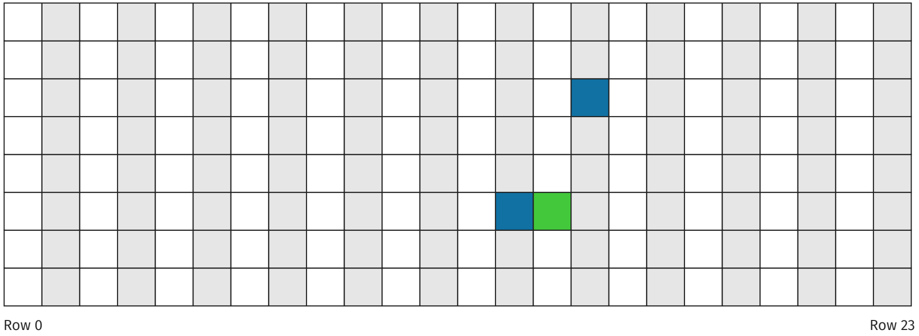
## Release page with flip



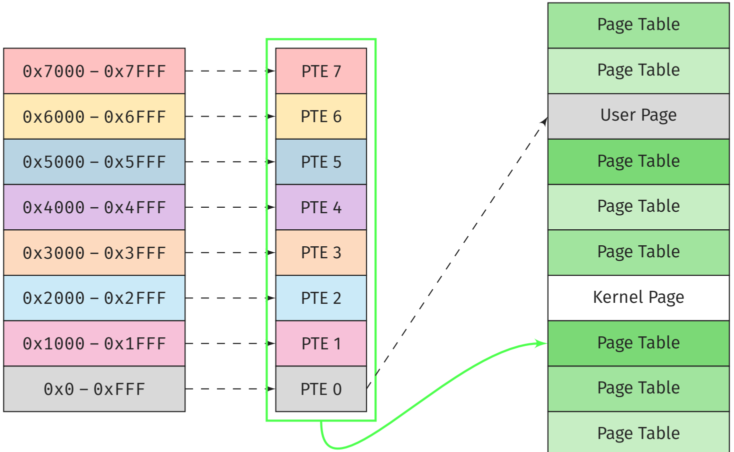
## Fill all remaining memory with page tables



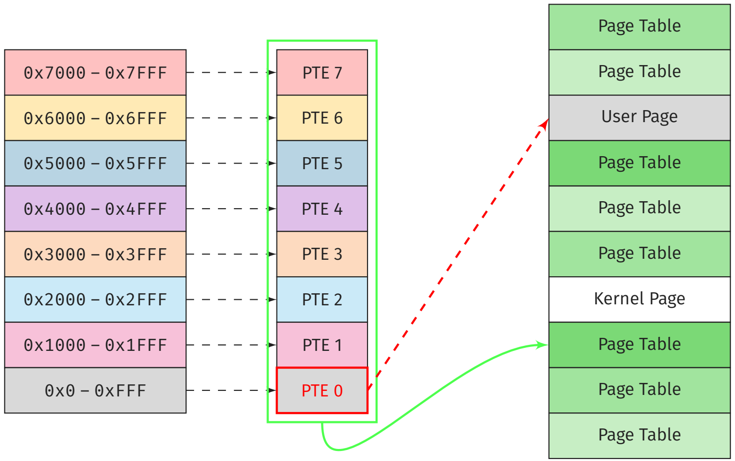
## Fill all remaining memory with page tables



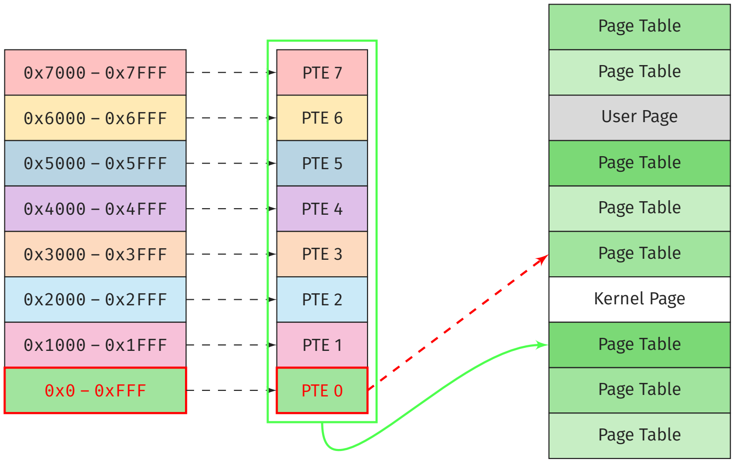
## Page Table Manipulation



## Page Table Manipulation

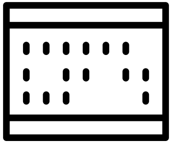


## Page Table Manipulation

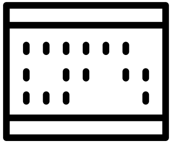


## Strategy: Flipping Page Table PPN bits

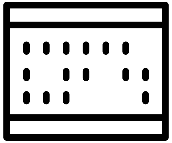
## Strategy: Flipping Page Table PPN bits



## Strategy: Flipping Page Table PPN bits

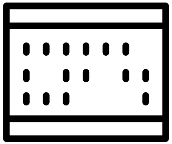


## Strategy: Flipping Page Table PPN bits



1. Scan for flips

## Strategy: Flipping Page Table PPN bits



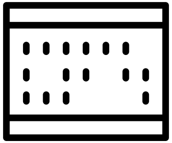
1. Scan for flips
2. Exhaust or massage memory to place a page table at target location

## Strategy: Flipping Page Table PPN bits

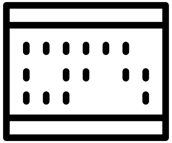
1. Scan for flips
2. Exhaust or massage memory to place a page table at target location
3. Gain access to your own page table $\rightarrow$ kernel privileges

## Flipping Page Table PPN bits

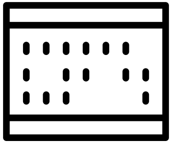
## Flipping Page Table PPN bits



## Flipping Page Table PPN bits

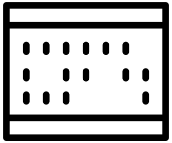


# Flipping Page Table PPN bits



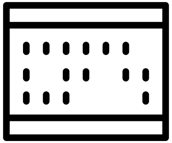
- Idea from [SD15]

## Flipping Page Table PPN bits



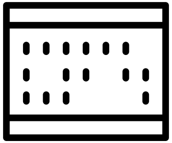
- Idea from [SD15]
- Same idea applied in several other works:

## Flipping Page Table PPN bits



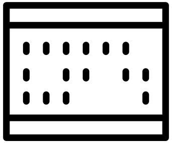
- Idea from [SD15]
- Same idea applied in several other works:
    - Rowhammer.js [GMM16]

## Flipping Page Table PPN bits



- Idea from [SD15]
- Same idea applied in several other works:
  - Rowhammer.js [GMM16]
  - One bit flips, one cloud flops [Xia+16]

## Flipping Page Table PPN bits



- Idea from [SD15]
- Same idea applied in several other works:
  - Rowhammer.js [GMM16]
  - One bit flips, one cloud flops [Xia+16]
  - Drammer [Vee+16]

## Post-Rowhammer Exploitation

## Post-Rowhammer Exploitation

## Post-Rowhammer Exploitation

## Post-Rowhammer Exploitation

- Scan entire physical memory (very fast) and:

## Post-Rowhammer Exploitation

- Scan entire physical memory (very fast) and:
    - Modify binary pages executed in root privileges [Xia+16]

## Post-Rowhammer Exploitation

- Scan entire physical memory (very fast) and:
  - Modify binary pages executed in root privileges [Xia+16]
  - Modify credential structs [Vee+16]

## Post-Rowhammer Exploitation

- Scan entire physical memory (very fast) and:
  - Modify binary pages executed in root privileges [Xia+16]
  - Modify credential structs [Vee+16]
  - Read keys [Xia+16]

## Post-Rowhammer Exploitation

- Scan entire physical memory (very fast) and:
  - Modify binary pages executed in root privileges [Xia+16]
  - Modify credential structs [Vee+16]
  - Read keys [Xia+16]
  - Corrupt signatures [BM16; Pod+18]

## Post-Rowhammer Exploitation

- Scan entire physical memory (very fast) and:
  - Modify binary pages executed in root privileges [Xia+16]
  - Modify credential structs [Vee+16]
  - Read keys [Xia+16]
  - Corrupt signatures [BM16; Pod+18]
  - Modify certificates

## Post-Rowhammer Exploitation

- Scan entire physical memory (very fast) and:
  - Modify binary pages executed in root privileges [Xia+16]
  - Modify credential structs [Vee+16]
  - Read keys [Xia+16]
  - Corrupt signatures [BM16; Pod+18]
  - Modify certificates
  - Configurations

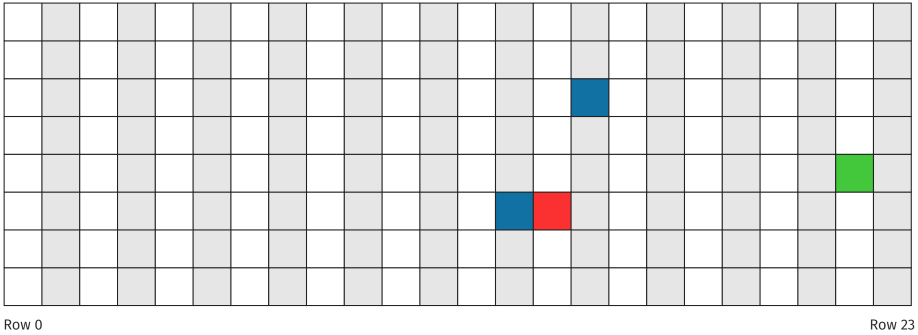
## Post-Rowhammer Exploitation

- Scan entire physical memory (very fast) and:
  - Modify binary pages executed in root privileges [Xia+16]
  - Modify credential structs [Vee+16]
  - Read keys [Xia+16]
  - Corrupt signatures [BM16; Pod+18]
  - Modify certificates
  - Configurations
  - etc.

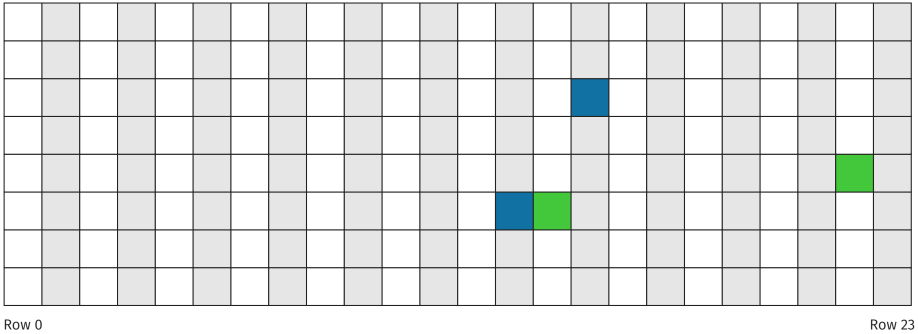
## Post-Rowhammer Exploitation

- Scan entire physical memory (very fast) and:
  - Modify binary pages executed in root privileges [Xia+16]
  - Modify credential structs [Vee+16]
  - Read keys [Xia+16]
  - Corrupt signatures [BM16; Pod+18]
  - Modify certificates
  - Configurations
  - etc.
- pages are pretty unique: 32768 bits per page

## Bit Flips + Page Deduplication

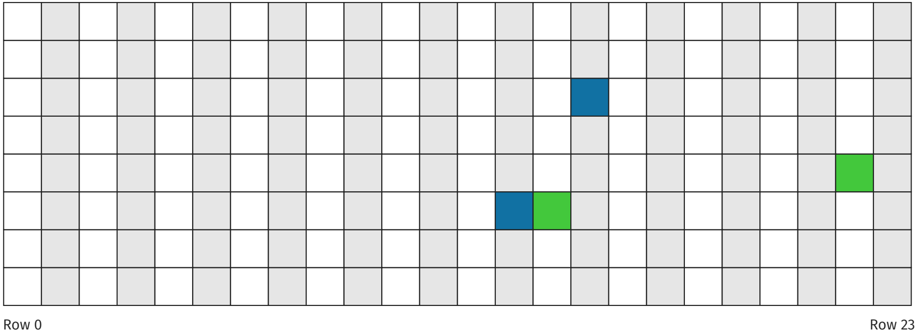


## Bit Flips + Page Deduplication



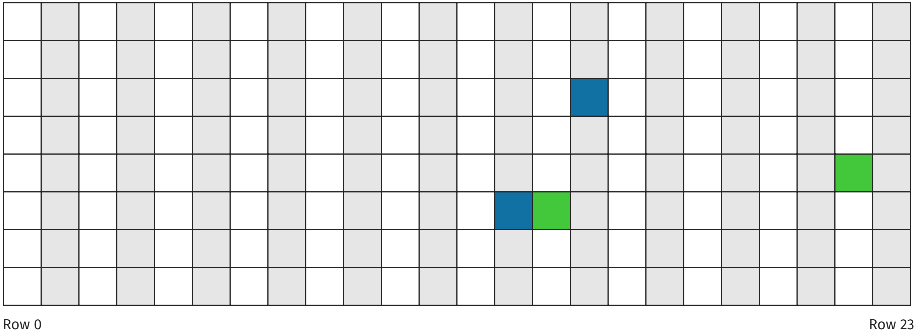
Page with bit flip is filled with target content

## Bit Flips + Page Deduplication



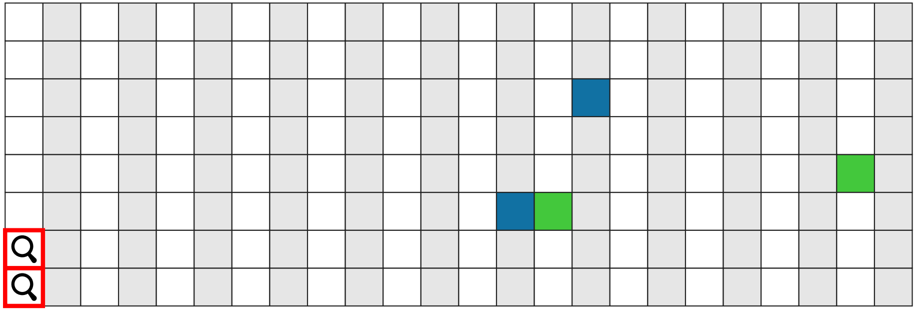
OS or hypervisor searches for duplicate pages

## Bit Flips + Page Deduplication



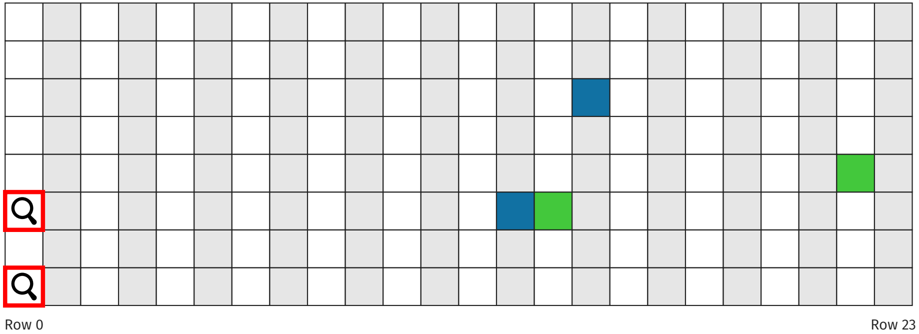
OS or hypervisor searches for duplicate pages

## Bit Flips + Page Deduplication



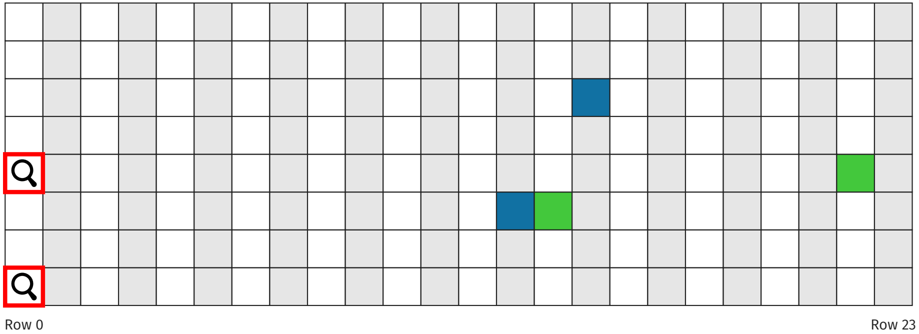
OS or hypervisor searches for duplicate pages

## Bit Flips + Page Deduplication



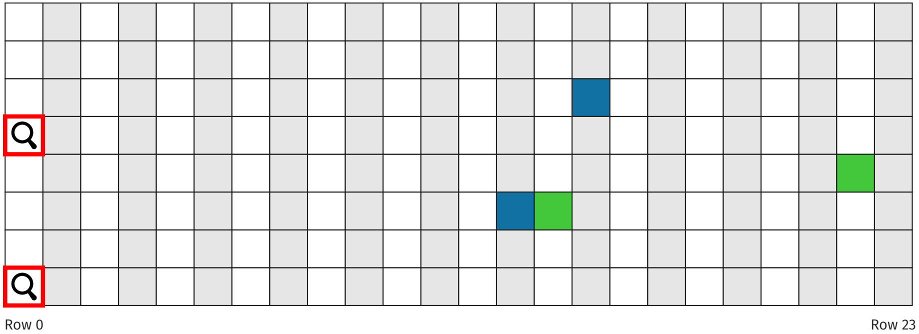
OS or hypervisor searches for duplicate pages

## Bit Flips + Page Deduplication



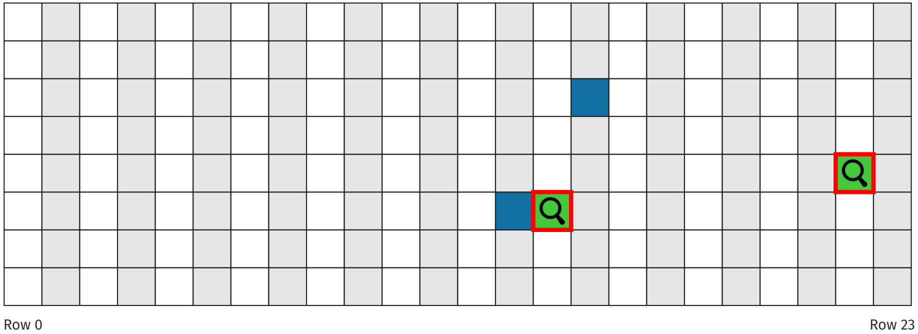
OS or hypervisor searches for duplicate pages

## Bit Flips + Page Deduplication



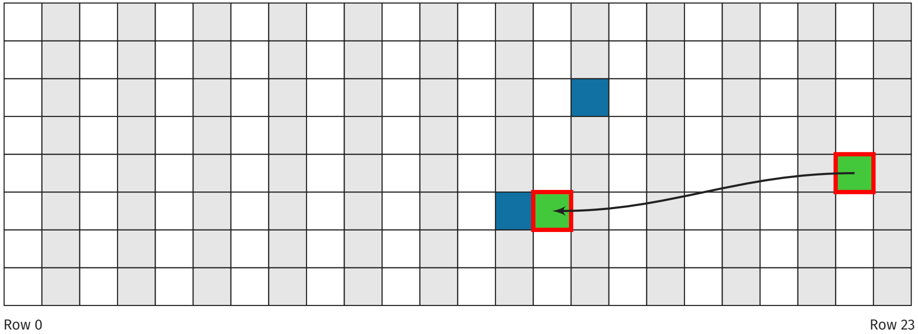
OS or hypervisor searches for duplicate pages

## Bit Flips + Page Deduplication



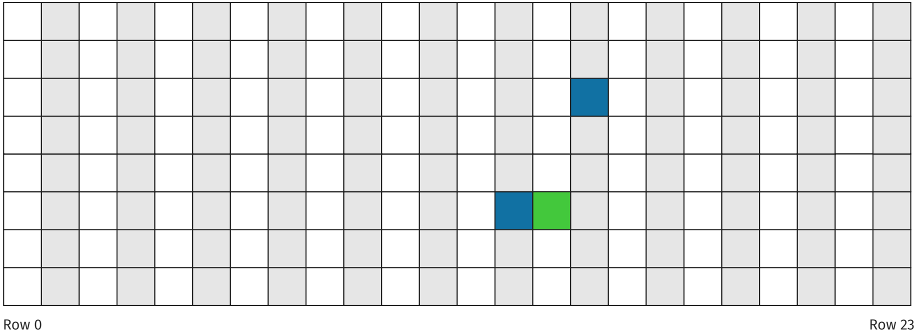
OS or hypervisor searches for duplicate pages

## Bit Flips + Page Deduplication



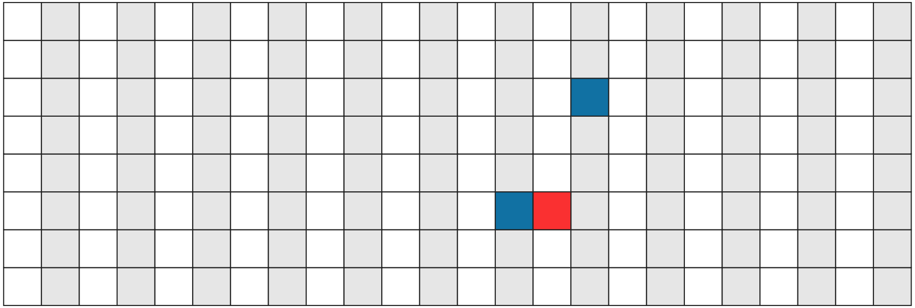
OS or hypervisor searches for duplicate pages

## Bit Flips + Page Deduplication



Hammer again + flip again

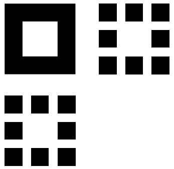
# Bit Flips + Page Deduplication


Row 0

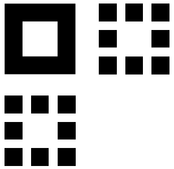
Row 23

## Strategy: Flipping in Deduplicated Pages

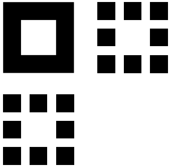
## Strategy: Flipping in Deduplicated Pages



## Strategy: Flipping in Deduplicated Pages

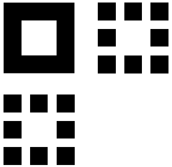


## Strategy: Flipping in Deduplicated Pages



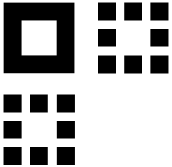
1. Scan for flips

## Strategy: Flipping in Deduplicated Pages



1. Scan for flips
2. Place content for deduplication so that flip can be exploited

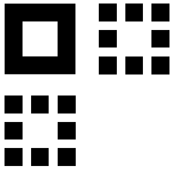
## Strategy: Flipping in Deduplicated Pages



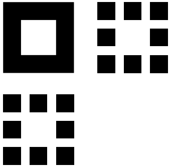
1. Scan for flips
2. Place content for deduplication so that flip can be exploited
3. Perform the bit change through Rowhammer

## Flipping in Deduplicated Pages

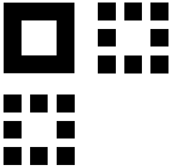
## Flipping in Deduplicated Pages



## Flipping in Deduplicated Pages

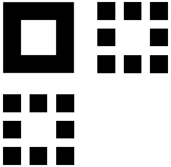


## Flipping in Deduplicated Pages



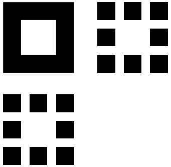
- Idea from [Bos+16]

## Flipping in Deduplicated Pages



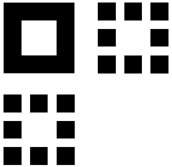
- Idea from [Bos+16]
  - Change data type (double $\rightarrow$ pointer)

## Flipping in Deduplicated Pages



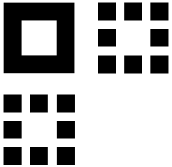
- Idea from [Bos+16]
  - Change data type (double $\rightarrow$ pointer)
  - Change pointer to good object $\rightarrow$ counterfeit object

# Flipping in Deduplicated Pages



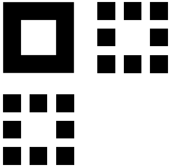
- Idea from [Bos+16]
  - Change data type (double $\rightarrow$ pointer)
  - Change pointer to good object $\rightarrow$ counterfeit object
- and from [Raz+16]

## Flipping in Deduplicated Pages



- Idea from [Bos+16]
    - Change data type (double $\rightarrow$ pointer)
    - Change pointer to good object $\rightarrow$ counterfeit object
- and from [Raz+16]
    - Corrupt authorized SSH keys

## Flipping in Deduplicated Pages

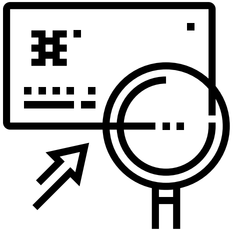


- Idea from [Bos+16]
  - Change data type (double $\rightarrow$ pointer)
  - Change pointer to good object $\rightarrow$ counterfeit object
- and from [Raz+16]
  - Corrupt authorized SSH keys
  - Corrupt Debian update URLs + RSA public key file

## How to mitigate Rowhammer?

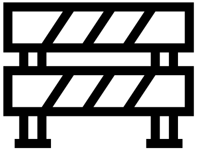
## Mitigations

Different mitigations have been proposed:



Detection

vs



Prevention

## Mitigations

Different mitigations have been proposed:

Software

**vs**

Hardware

## Mitigations

Different mitigations have been proposed:

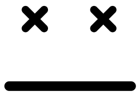
Short Term

**vs**

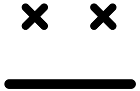
Long Term

## Quick fixes

- No `clflush` instruction

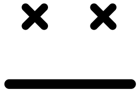


## Quick fixes



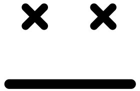
- No `clflush` instruction → Rowhammer.js

## Quick fixes

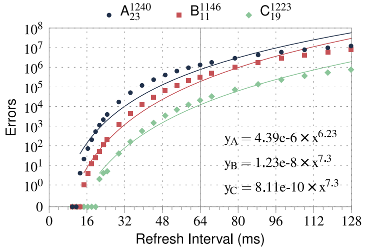


- No `clflush` instruction $\rightarrow$ Rowhammer.js
- Increase the refresh rate

## Quick fixes

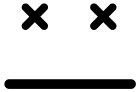


- No `clflush` instruction → Rowhammer.js
- Increase the refresh rate
  → Would need to be increased by 7× to eliminate all bit flips

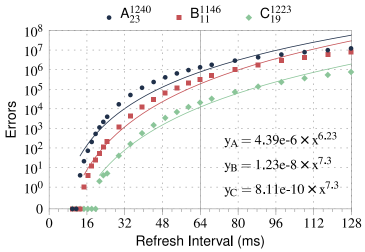


Errors depending on refresh interval [Kim+14]

## Quick fixes



- No `clflush` instruction → Rowhammer.js
- 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]

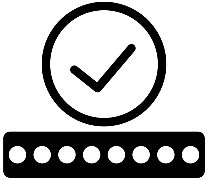
## What about ECC?

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

## What about ECC?

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

## What about ECC?



- ECC protection: server can handle or correct single bit errors
- No standard for event reporting
- In practice [Lan16]
  - Common: server counts ECC errors and report only if they reach a threshold (e.g., $>$ 100 bit flips / hour)

## What about ECC?

- ECC protection: server can handle or correct single bit errors
- No standard for event reporting
- In practice [Lan16]
  - Common: server counts ECC errors and report only if they reach a threshold (e.g., $>$ 100 bit flips / hour)
  - Some server vendors never report errors to the OS

## What about ECC?

- ECC protection: server can handle or correct single bit errors
- No standard for event reporting
- In practice [Lan16]
  - 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

## Preventing Rowhammer attacks in hardware (1/3)

## Preventing Rowhammer attacks in hardware (1/3)

## Preventing Rowhammer attacks in hardware (1/3)

Original ideas from [Kim+14]

## Preventing Rowhammer attacks in hardware (1/3)

Original ideas from [Kim+14]

- Making better DRAM chips that are not vulnerable

## 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)

## 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

## 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

## 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]

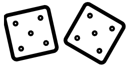
- Making better DRAM chips that are not vulnerable
- Using error correcting codes (ECC)
- Increasing the refresh rate
- Remapping/retiring faulty cells after manufacturing
- Identifying hammered rows at runtime and refreshing neighbors

→ Expensive, performance overhead, or increased power consumption

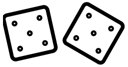
## Preventing Rowhammer attacks in hardware (2/3)

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)



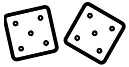
PARA - Probabilistic Adjacent Row Activation [Kim+14]

- 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)

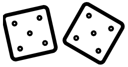
PARA - Probabilistic Adjacent Row Activation [Kim+14]

- 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)

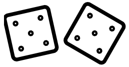
PARA - Probabilistic Adjacent Row Activation [Kim+14]

- 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)

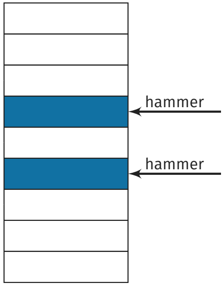
PARA - Probabilistic Adjacent Row Activation [Kim+14]

- 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 $9.4 \times 10^{-14}$



## Preventing Rowhammer attacks in hardware (3/3)

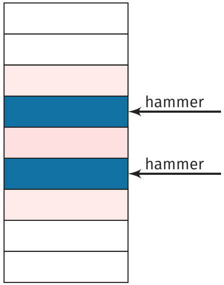
Target Row Refresh (TRR)



## Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

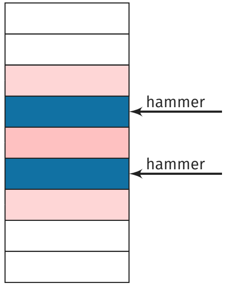
- Counter per row



## Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

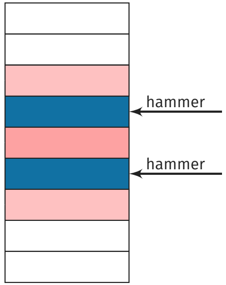
- Counter per row
- Increment neighbor rows



## Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

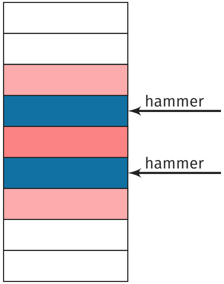
- Counter per row
- Increment neighbor rows



## Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

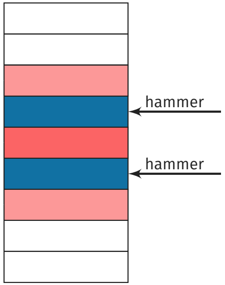
- Counter per row
- Increment neighbor rows



## Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

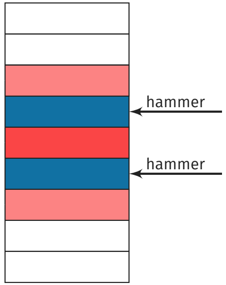
- Counter per row
- Increment neighbor rows



## Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

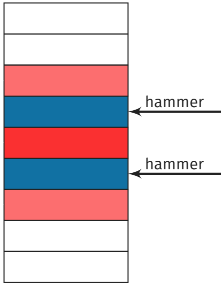
- Counter per row
- Increment neighbor rows



## Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

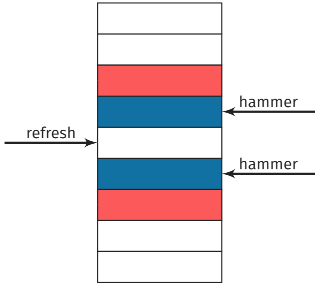
- Counter per row
- Increment neighbor rows



## Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

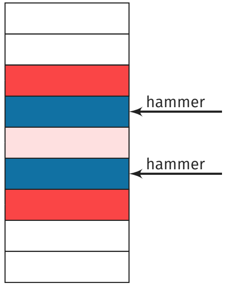
- Counter per row
- Increment neighbor rows
- Refresh when counter reaches a threshold



## Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

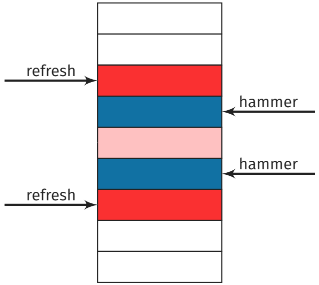
- Counter per row
- Increment neighbor rows
- Refresh when counter reaches a threshold



## Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

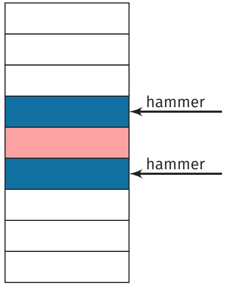
- Counter per row
- Increment neighbor rows
- Refresh when counter reaches a threshold



## Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

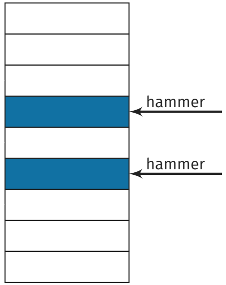
- Counter per row
- Increment neighbor rows
- Refresh when counter reaches a threshold



## Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

- Counter per row
- Increment neighbor rows
- Refresh when counter reaches a threshold



We flipped bits on DDR4 with TRR activated!

## Preventing Rowhammer attacks in software

“nohammer” kernel module [Cor16]

## Preventing Rowhammer attacks in software

"nohammer" kernel module [Cor16]

- Refresh rate of 8 ms would prevent Rowhammer on most systems

## Preventing Rowhammer attacks in software

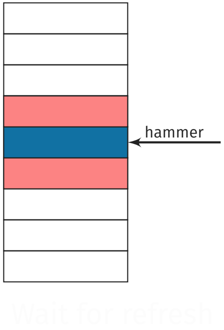
“nohammer” kernel module [Cor16]

- Refresh rate of 8 ms would prevent Rowhammer on most systems
- Use PMC to measure cache misses per 64 ms interval

## Preventing Rowhammer attacks in software

“nohammer” kernel module [Cor16]

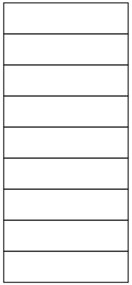
- Refresh rate of 8 ms would prevent Rowhammer on most systems
- Use PMC to measure cache misses per 64 ms interval
- Limit cache miss rate to 1/8 of maximum



## Preventing Rowhammer attacks in software

“nohammer” kernel module [Cor16]

- Refresh rate of 8 ms would prevent Rowhammer on most systems
- Use PMC to measure cache misses per 64 ms interval
- Limit cache miss rate to 1/8 of maximum



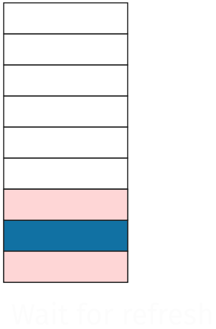
Wait for refresh

Wait for refresh

## Preventing Rowhammer attacks in software

"nohammer" kernel module [Cor16]

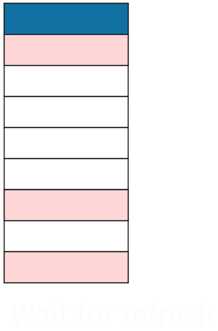
- Refresh rate of 8 ms would prevent Rowhammer on most systems
- Use PMC to measure cache misses per 64 ms interval
- Limit cache miss rate to 1/8 of maximum



## Preventing Rowhammer attacks in software

"nohammer" kernel module [Cor16]

- Refresh rate of 8 ms would prevent Rowhammer on most systems
- Use PMC to measure cache misses per 64 ms interval
- Limit cache miss rate to 1/8 of maximum



## Preventing Rowhammer attacks in software

“nohammer” kernel module [Cor16]

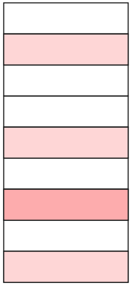
- Refresh rate of 8 ms would prevent Rowhammer on most systems
- Use PMC to measure cache misses per 64 ms interval
- Limit cache miss rate to 1/8 of maximum

Wait for refresh

## Preventing Rowhammer attacks in software

"nohammer" kernel module [Cor16]

- Refresh rate of 8 ms would prevent Rowhammer on most systems
- Use PMC to measure cache misses per 64 ms interval
- Limit cache miss rate to 1/8 of maximum



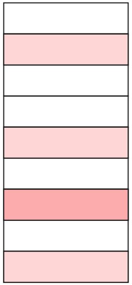
Wait for refresh

Wait for refresh

## Preventing Rowhammer attacks in software

"nohammer" kernel module [Cor16]

- Refresh rate of 8 ms would prevent Rowhammer on most systems
- Use PMC to measure cache misses per 64 ms interval
- Limit cache miss rate to 1/8 of maximum

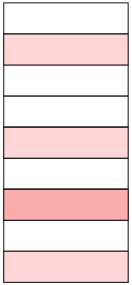


Performance?

## Preventing Rowhammer attacks in software

"nohammer" kernel module [Cor16]

- Refresh rate of 8 ms would prevent Rowhammer on most systems
- Use PMC to measure cache misses per 64 ms interval
- Limit cache miss rate to 1/8 of maximum

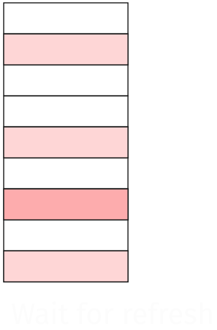


Performance? Grand Pwning Unit [Fri+18],

## Preventing Rowhammer attacks in software

"nohammer" kernel module [Cor16]

- Refresh rate of 8 ms would prevent Rowhammer on most systems
- Use PMC to measure cache misses per 64 ms interval
- Limit cache miss rate to 1/8 of maximum

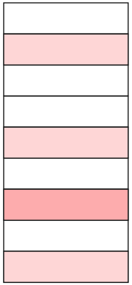


Performance? Grand Pwning Unit [Fri+18], ThrowHammer [Tat+18],

## Preventing Rowhammer attacks in software

“nohammer” kernel module [Cor16]

- Refresh rate of 8 ms would prevent Rowhammer on most systems
- Use PMC to measure cache misses per 64 ms interval
- Limit cache miss rate to 1/8 of maximum



Performance? Grand Pwning Unit [Fri+18], ThrowHammer [Tat+18], NetHammer [Lip+17].

## Preventing Rowhammer attacks in software

MASCAT - Stopping Microarchitectural Attacks Before Execution [IES17]

- Static analysis of the binary

## Preventing Rowhammer attacks in software

MASCAT - Stopping Microarchitectural Attacks Before Execution [IES17]

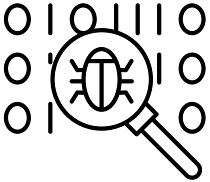
- Static analysis of the binary
- Detect suspicious instruction sequences (`clflush`, `rdtsc`, fences, ...)
- Open problem: false positives

## Preventing Rowhammer attacks in software

MASCAT - Stopping Microarchitectural Attacks Before Execution [IES17]

- Static analysis of the binary
- Detect suspicious instruction sequences (`clflush`, `rdtsc`, fences, ...)
- Open problem: false positives

ThrowHammer [Tat+18], NetHammer [Lip+17].



## Preventing Rowhammer attacks in software

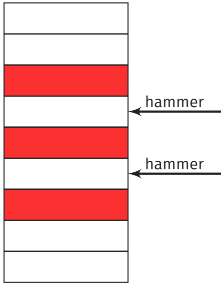
ANVIL [Awe+16]

- Uses performance counters to detect rowhammer

## Preventing Rowhammer attacks in software

ANVIL [Awe+16]

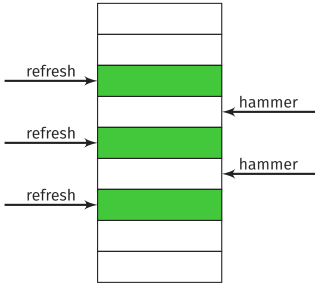
- Uses performance counters to detect rowhammer
- Activate rows neighbor rows to prevent flips
- Similar as PARA, but in software



## Preventing Rowhammer attacks in software

ANVIL [Awe+16]

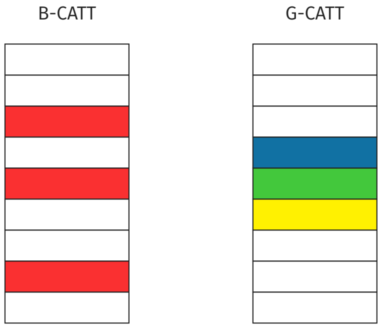
- Uses performance counters to detect rowhammer
- Activate rows neighbor rows to prevent flips
- Similar as PARA, but in software



What if performance counters do not work? [Gru+18; Jan+17]

## Preventing Rowhammer attacks in software

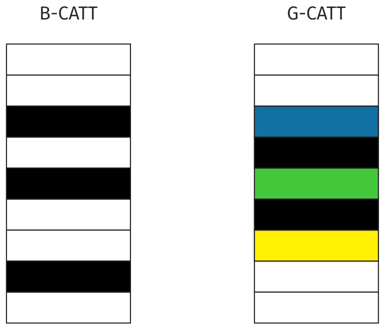
- B-CATT: disable vulnerable physical memory [Bra+17]
- G-CATT: isolate security domains in physical memory based on potential vulnerability [Bra+17]



# Preventing Rowhammer attacks in software

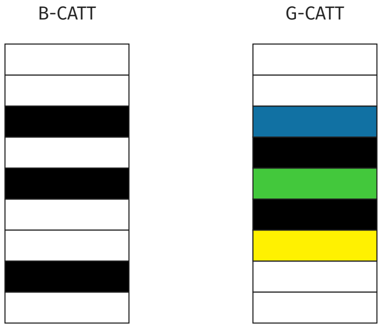
- B-CATT: disable vulnerable physical memory [Bra+17]
- G-CATT: isolate security domains in physical memory based on potential vulnerability [Bra+17]

B-CATT: Might block 95% of RAM [Gru+18; Vee+18]



## Preventing Rowhammer attacks in software

- B-CATT: disable vulnerable physical memory [Bra+17]
- G-CATT: isolate security domains in physical memory based on potential vulnerability [Bra+17]

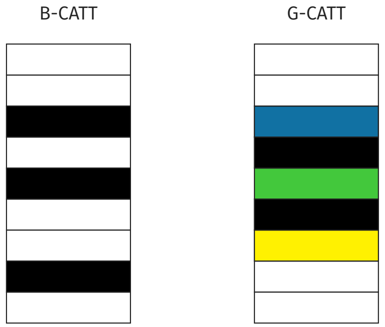


B-CATT: Might block 95% of RAM [Gru+18; Vee+18]

G-CATT: What about non-kernel or shared pages? [Gru+18; CZN18]

## Preventing Rowhammer attacks in software

- B-CATT: disable vulnerable physical memory [Bra+17]
- G-CATT: isolate security domains in physical memory based on potential vulnerability [Bra+17]



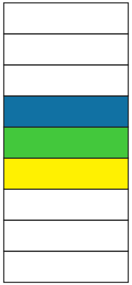
B-CATT: Might block 95% of RAM [Gru+18; Vee+18]

G-CATT: What about non-kernel or shared pages? [Gru+18; CZN18]

G-CATT: Bit flips more than 8 “rows” apart [Kim+14; Gru+18]

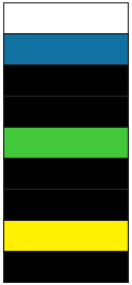
## GuardION

- Isolate DMA buffers in physical memory [Vee+18]



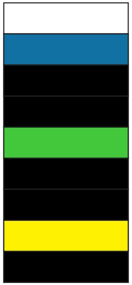
## GuardION

- Isolate DMA buffers in physical memory [Vee+18]



## GuardION

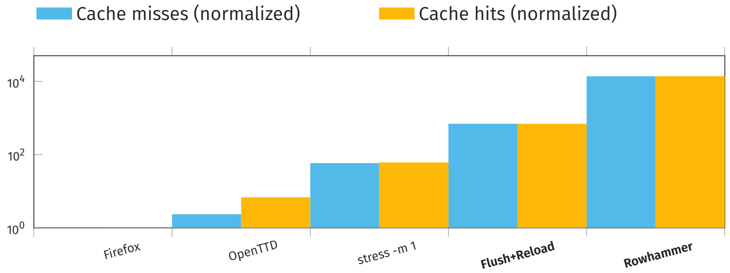
- Isolate DMA buffers in physical memory [Vee+18]



Bit flips more than 8 “rows” apart [Kim+14; Gru+18]

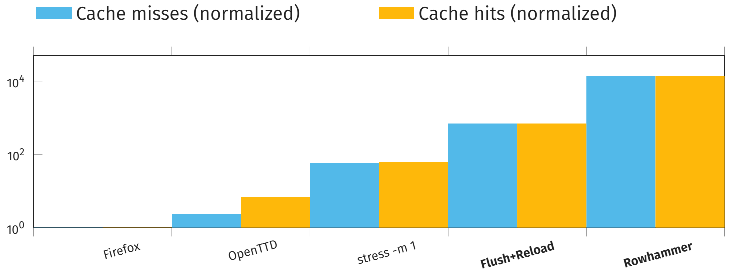
## Detecting Rowhammer attacks

- Rowhammer: lots of cache misses that can be monitored with hardware performance counters [HF15; Gru+16; CSY15; Pay16]



# Detecting Rowhammer attacks

- Rowhammer: lots of cache misses that can be monitored with hardware performance counters [HF15; Gru+16; CSY15; Pay16]



What if performance counters do not work because we run in SGX? [Gru+18; Jan+17]

## Defenses Overview

| Methodology \ Defense | MASCAT | Chiapetta et al. [CSY15] | CloudRadar | Herath and Fogh [HF15] | HexPADS | perf | ANVIL | nohammer | No OOM | G-CATT | B-CATT | TRR | MAC | PARA/CRA/PRA | ARMOR | ECC/Chipkill | Refresh Rate |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Detection** | | | | | | | | | | | | | | | | | |
| Static Analysis | ● | ○ | ○ | ◐ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Performance Counters | ○ | ● | ● | ● | ● | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Memory Access Pattern | ○ | ○ | ○ | ○ | ○ | ○ | ● | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| **Neutralization** | | | | | | | | | | | | | | | | | |
| Physical Proximity | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Memory Footprint | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| **Elimination** | | | | | | | | | | | | | | | | | |
| Bootloader | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ |
| Hardware Modification | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ● | ● | ● | ● | ○ |
| BIOS Update | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● |

## What if you don’t need to hammer two or more rows?

## What if you don’t need to hammer two or more rows?

## One-location hammering

## How to hammer?

- There are two different hammering techniques

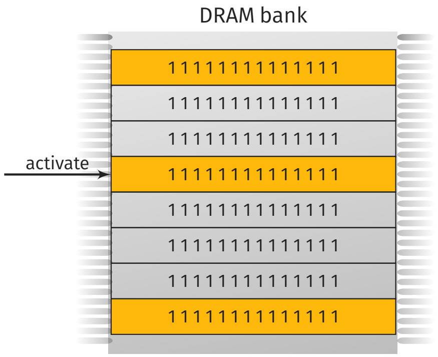
## How to hammer?

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

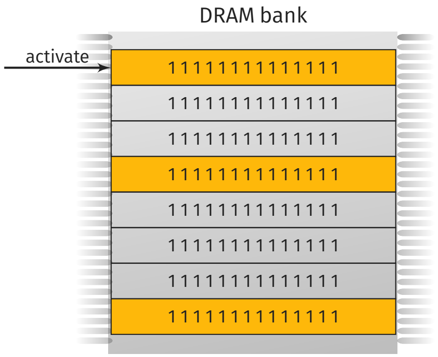
## How to hammer?

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

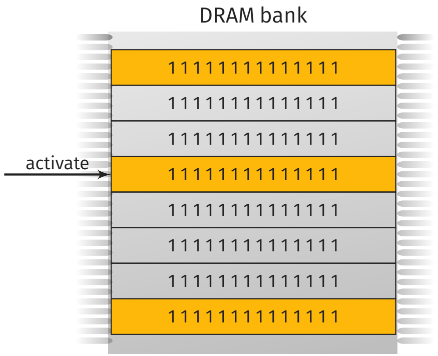
## #1 - Single-sided hammering



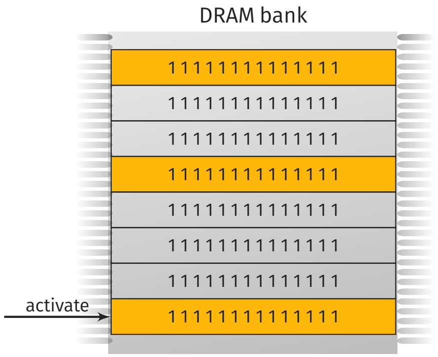
## #1 - Single-sided hammering



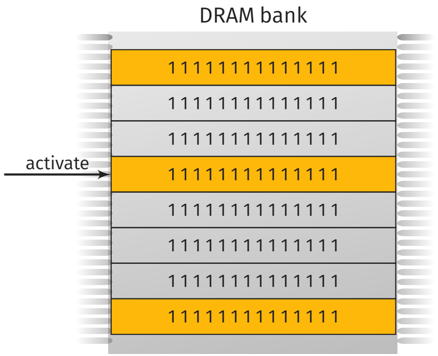
## #1 - Single-sided hammering



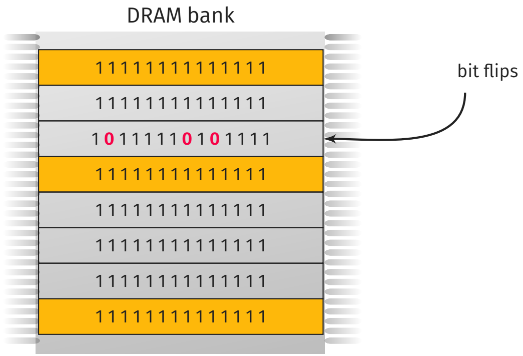
## #1 - Single-sided hammering



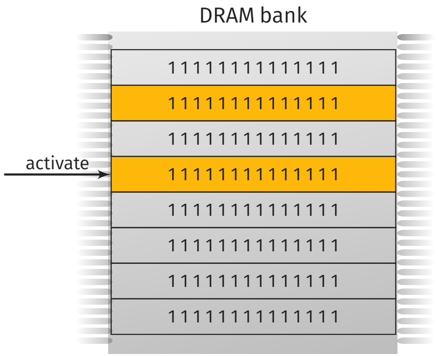
## #1 - Single-sided hammering



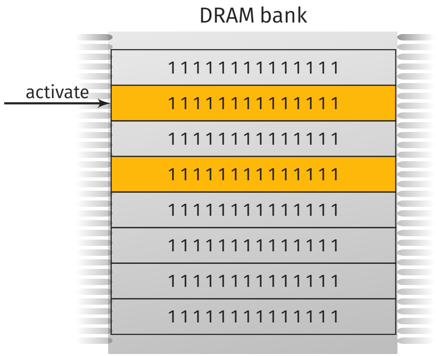
## #1 - Single-sided hammering



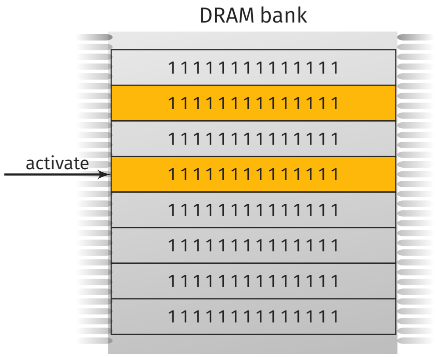
## #2 - Double-sided hammering



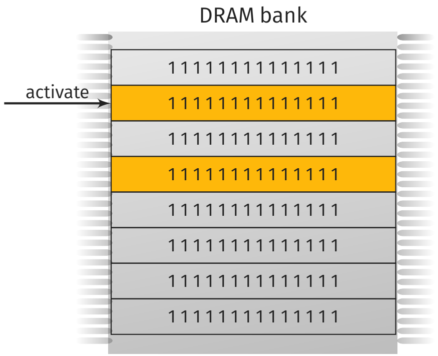
## #2 - Double-sided hammering



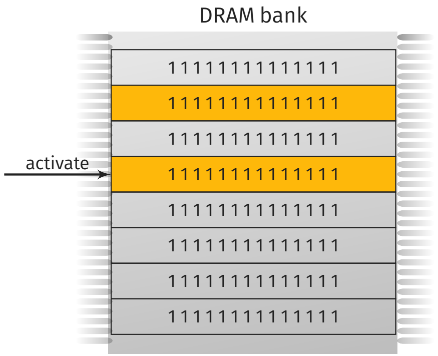
## #2 - Double-sided hammering



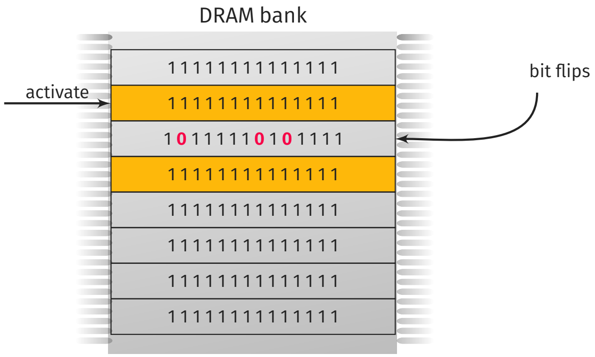
## #2 - Double-sided hammering



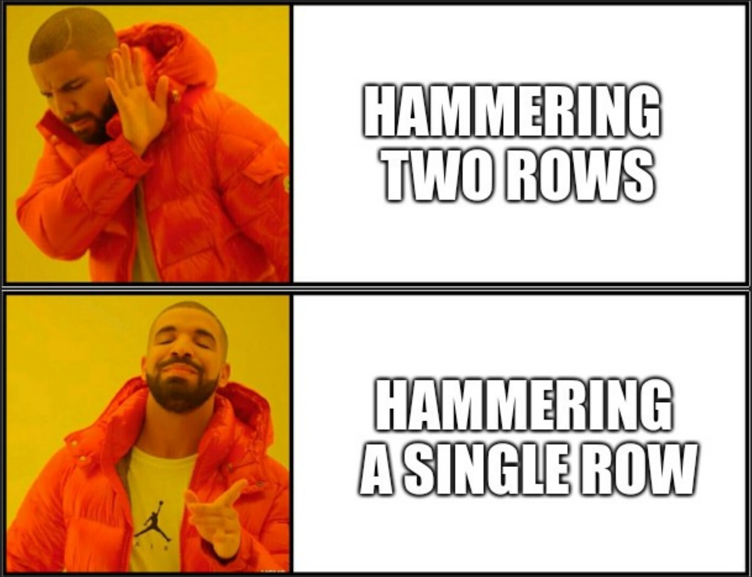
## #2 - Double-sided hammering



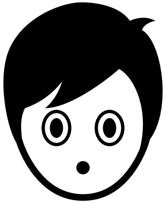
# #2 - Double-sided hammering



HAMMERING TWO ROWS

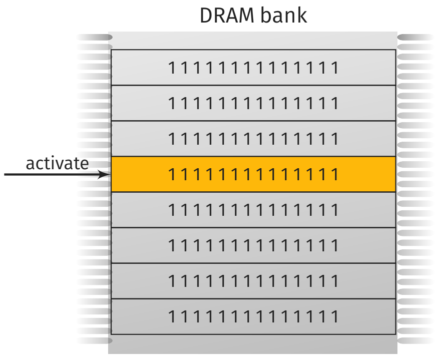

HAMMERING TWO ROWS
HAMMERING A SINGLE ROW

## Hammering techniques

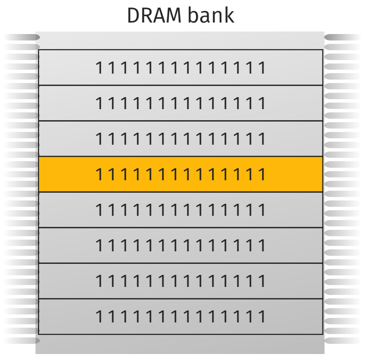


- There are three different hammering techniques
- #1: Hammer one row next to victim row and other random rows
- #2: Hammer two rows neighboring victim row
- #3: Hammer only one row next to victim row

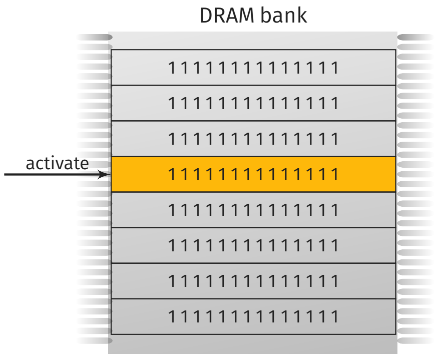
## #3 - One-location hammering



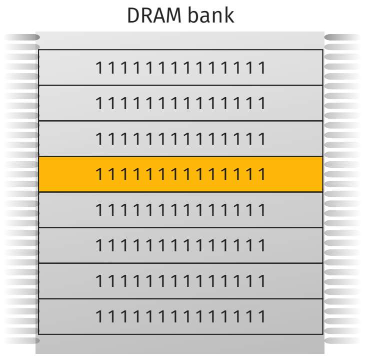
## #3 - One-location hammering



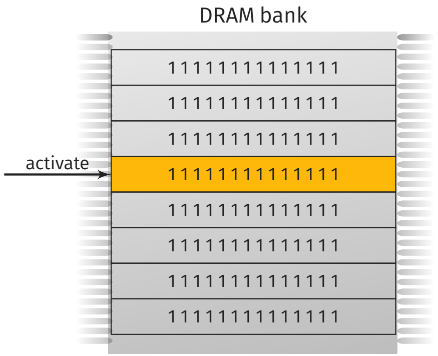
## #3 - One-location hammering



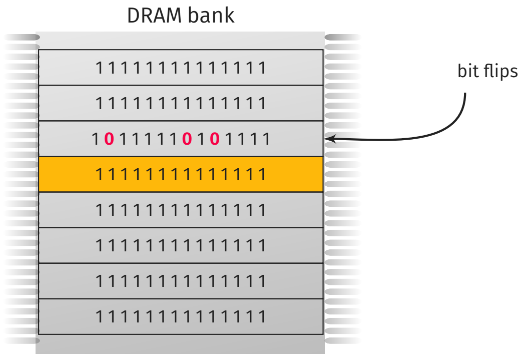
## #3 - One-location hammering

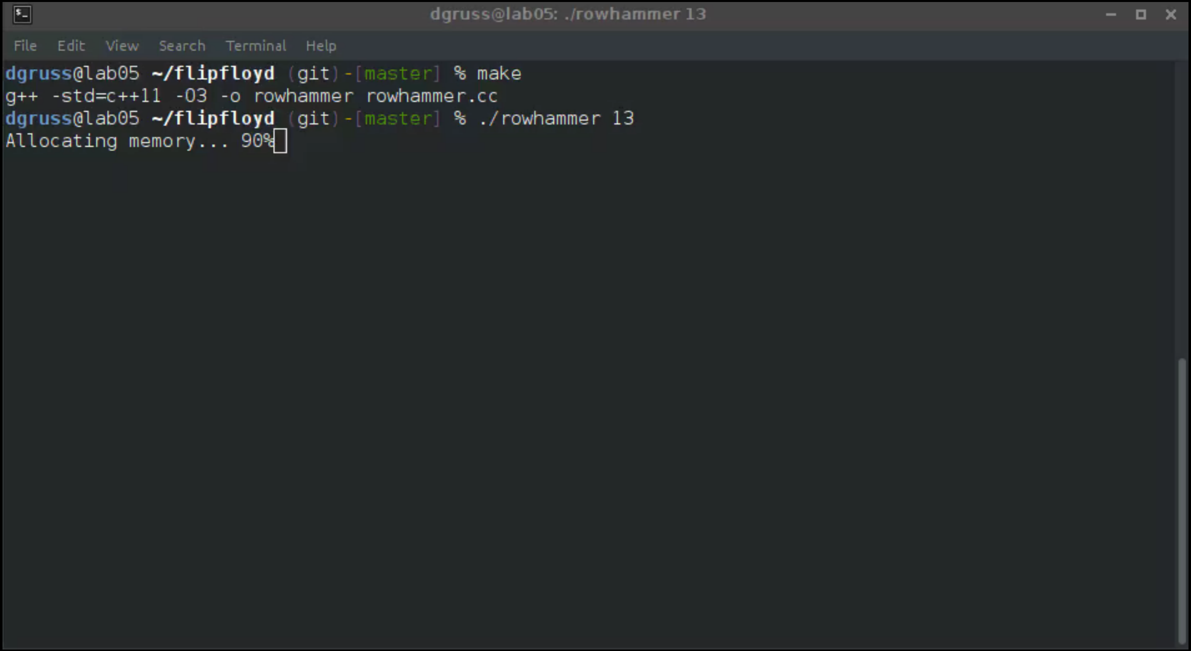


## #3 - One-location hammering



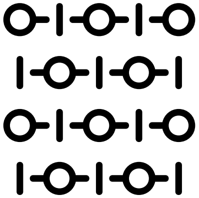
## #3 - One-location hammering




dgruss@lab05: ./rowhammer 13
File Edit View Search Terminal Help

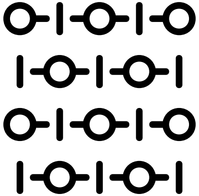
```
dgruss@lab05 ~/flipfloyd (git)-[master] % make
g++ -std=c++11 -O3 -o rowhammer rowhammer.cc
dgruss@lab05 ~/flipfloyd (git)-[master] % ./rowhammer 13
Allocating memory... 90%
```

## Memory-Controller Policies



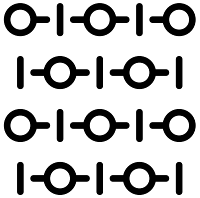
- **Open-page policy**: Keep row opened and buffered

## Memory-Controller Policies



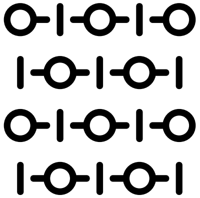
- **Open-page policy**: Keep row opened and buffered
  - Low latency for subsequent accesses to same row
  - High latency for accesses to any other row

## 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

## 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

## 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 [Dav+11]

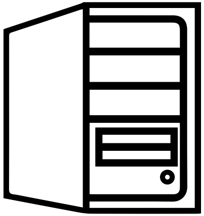
## Memory-Controller Policies

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

## Memory-Controller Policies

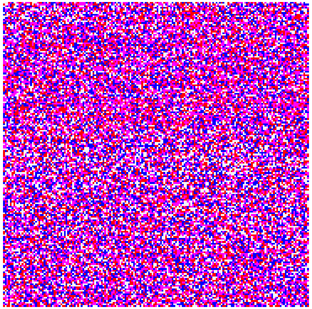
- Policies that preemptively close rows, would allow one-location hammering
- We observed close-page policies on desktop computers

## Memory-Controller Policies



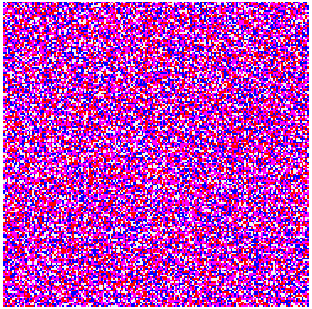
- Policies that preemptively close rows, would allow one-location hammering
- We observed close-page policies on desktop computers
- Mobile devices (e.g., laptops) seem to use mostly open-page policies

## How well does it work?


**Double-sided**

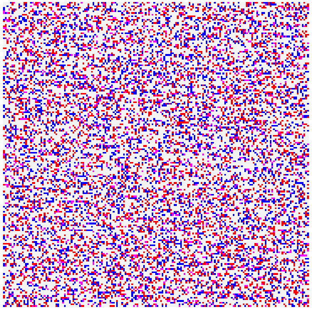
77.0 % bit offsets

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

## What if we cannot target kernel pages?

## What if we cannot target kernel pages?

## Opcode Flipping

## 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

## Opcode Flipping

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

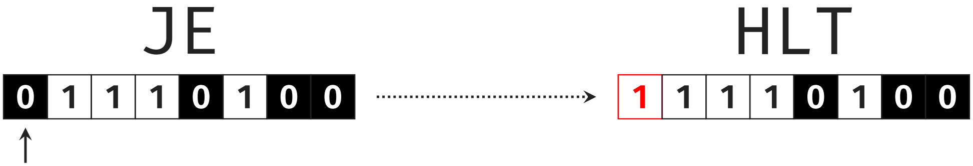
## Opcode Flipping

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

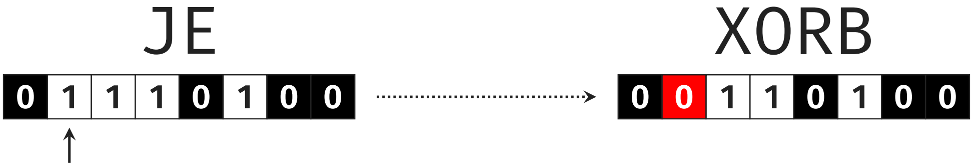
## Opcode Flipping

- 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` (easy to exploit)

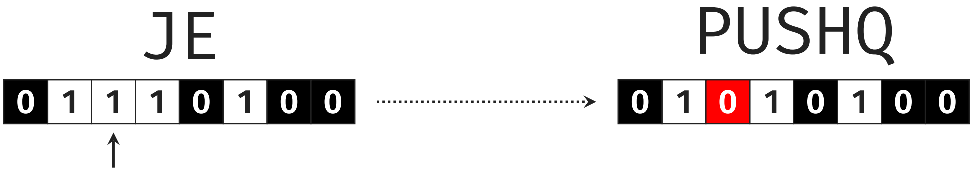
## Opcode Flipping - Conditional Jump



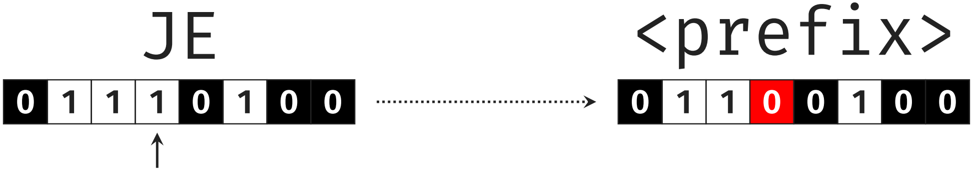
## Opcode Flipping - Conditional Jump



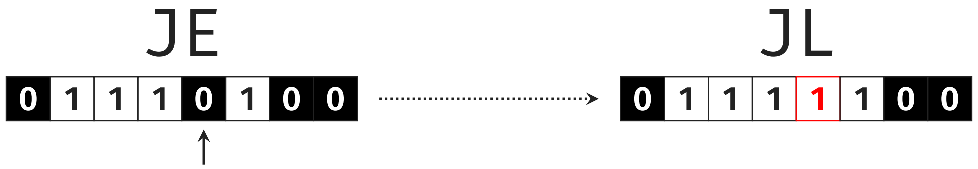
# Opcode Flipping - Conditional Jump



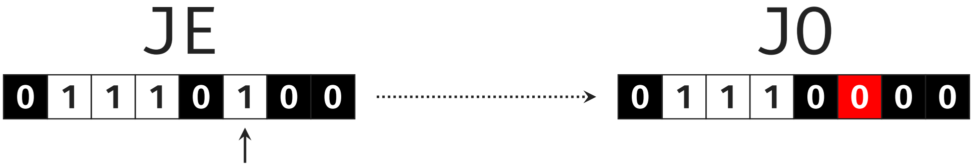
## Opcode Flipping - Conditional Jump



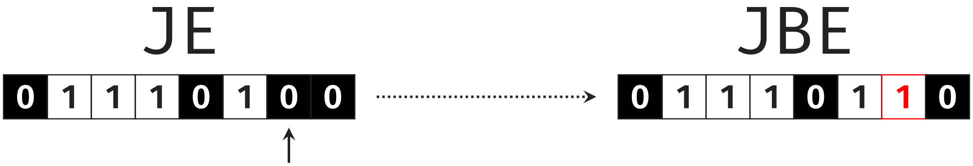
## Opcode Flipping - Conditional Jump



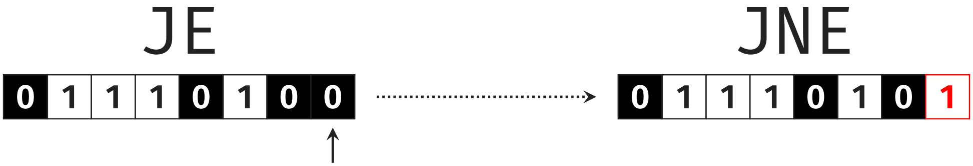
# Opcode Flipping - Conditional Jump



# 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 bit flips

## Opcode Flipping

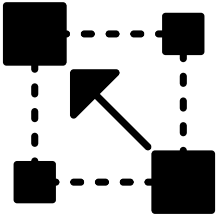
- Conditional jumps are not the only targets
- Other targets include
    - Comparisons
    - Addresses of memory loads/stores
    - Address calculations
    - ...
- Manual analysis of sudo revealed 29 possible bit flips
- They all somehow skipped the password check

## How to get the target virtual page to the target physical location?

## How to get the target virtual page to the target physical location?

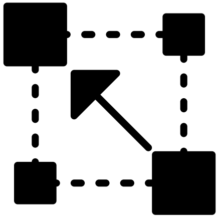
## Memory Waylaying

## Placing the binary



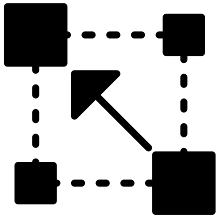
- Not as easy as with page tables

## Placing the binary



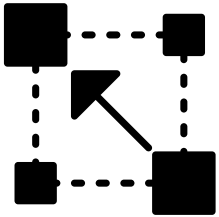
- Not as easy as with page tables
- Binary only once in memory + stays in memory (in the page cache) even after termination

## Placing the binary



- Not as easy as with page tables
- Binary only once in memory + stays in memory (in the page cache) even after termination
- Only evicted if page cache is full

## Placing the binary



- Not as easy as with page tables
- Binary only once in memory + stays in memory (in the page cache) even after termination
- Only evicted if page cache is full
- Page cache usually occupies all unused memory

## Page Cache

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

## Page Cache

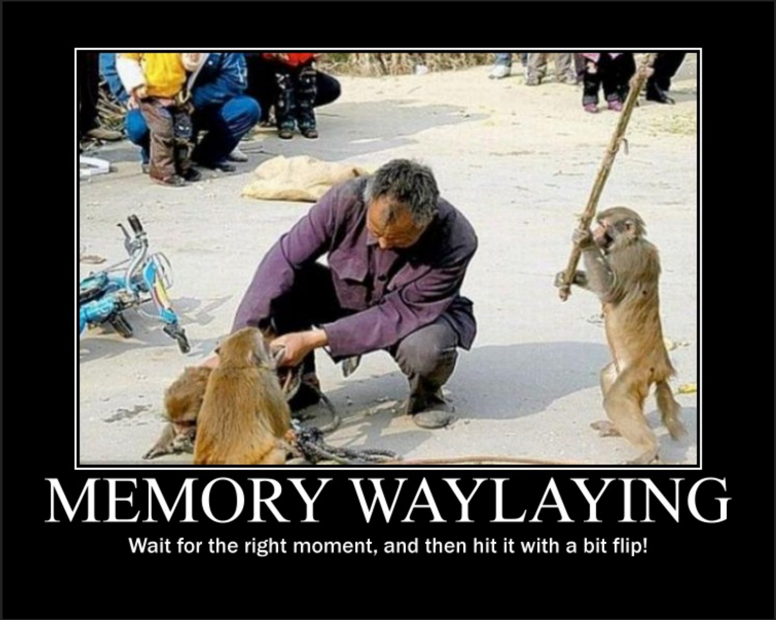
- If a binary is loaded the first time, it is loaded to the memory
- It stays in memory (in the page cache) even after execution

## 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

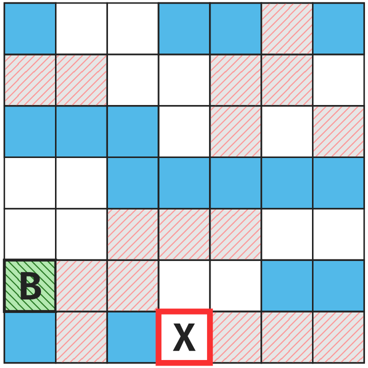
## Page Cache

- 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

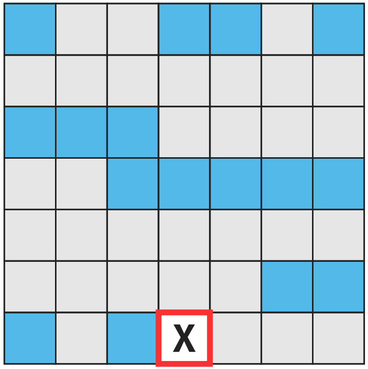

MEMORY WAYLAYING
Wait for the right moment, and then hit it with a bit flip!

Memory Waylaying

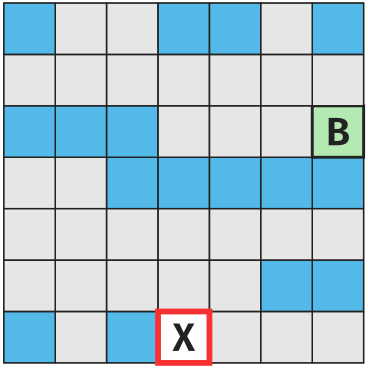
## (1) Start



## (2) Evict Page Cache

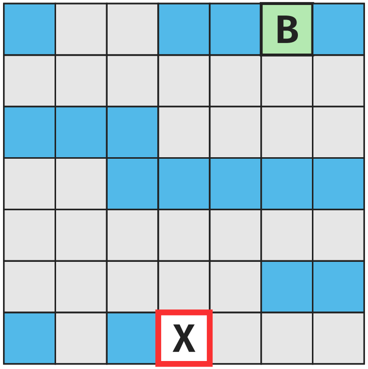


## (3) Access Binary

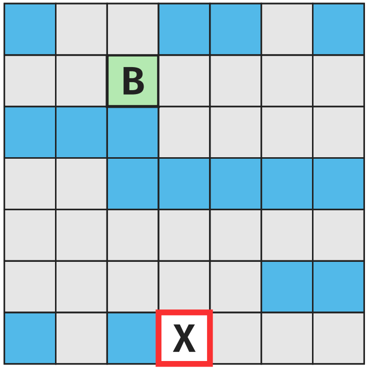


## Memory Waylaying

### (4) Evict + Access

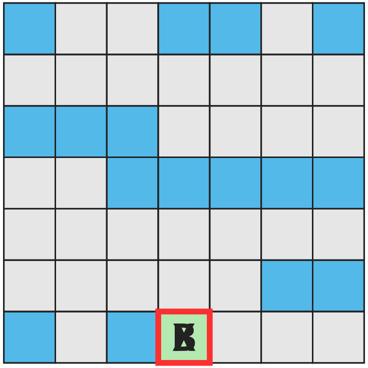


## (5) Evict + Access



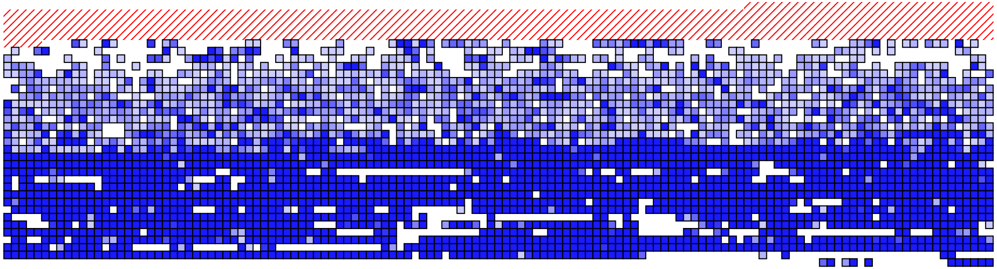
## Memory Waylaying

(6) Stop if target reached



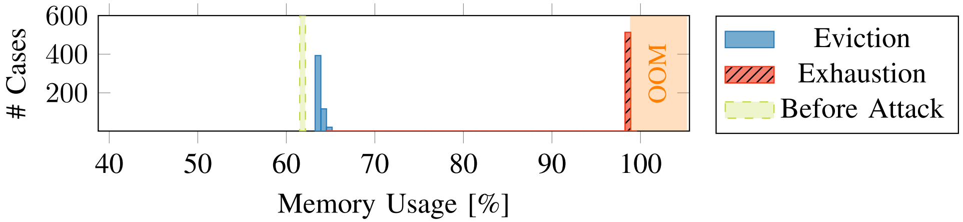
## How well does it work?

- New pages cover most of the physical memory



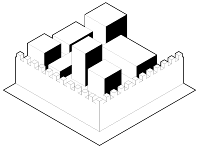
## How well does it work?

- Great advantage over memory massaging: only negligible memory footprint



## Rowhammer + SGX = Cheap Denial of Service

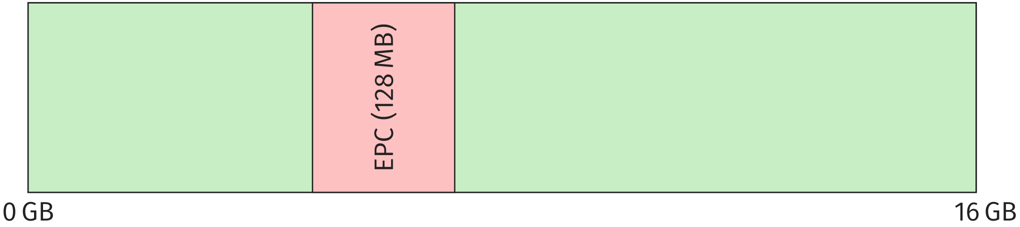
## Intel SGX



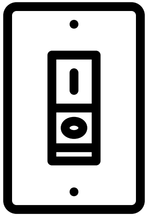
- Instruction-set extension
- Integrity and confidentiality of code and data in untrusted environments
- Run with user privileges and restricted, e.g., no system calls
- Run programs in enclaves using protected areas of memory

## SGX Encrypted Memory

## SGX Encrypted Memory



## Bit Flips in the EPC



- What happens if a bit flips in the EPC?

## Bit Flips in the EPC

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

## Bit Flips in the EPC

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

→ Locks up the memory controller

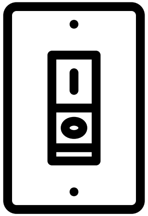
## Bit Flips in the EPC

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

→ Locks up the memory controller

→ Not a single further memory access!

## Bit Flips in the EPC



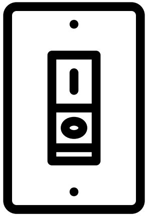
- What happens if a bit flips in the EPC?
- Integrity check will fail!

→ Locks 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!

→ Locks up the memory controller

→ Not a single further memory access!

→ System halts immediately

SOUNDS UNSAFE?
IT IS UNSAFE!
imgflip.com

## Bit Flips in the EPC

- If a malicious enclave induces a bit flip, ...

## Bit Flips in the EPC

- If a malicious enclave induces a bit flip, ...
- ...the entire machine halts

## Bit Flips in the EPC

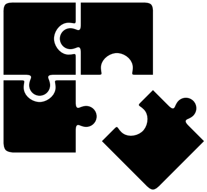
- If a malicious enclave induces a bit flip, ...
- ...the entire machine halts
- ...including co-located tenants

## Bit Flips in the EPC

- If a malicious enclave induces a bit flip, ...
- ...the entire machine halts
- ...including co-located tenants
- Denial-of-Service Attacks in the Cloud [Gru+18; Jan+17]

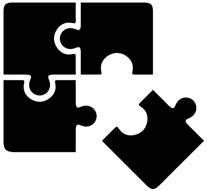
## SGX + One-location Hammering + Opcode Flipping = Undetectable Exploit

## (Ab)using SGX Protection



- SGX protects software from malicious environments

## (Ab)using SGX Protection

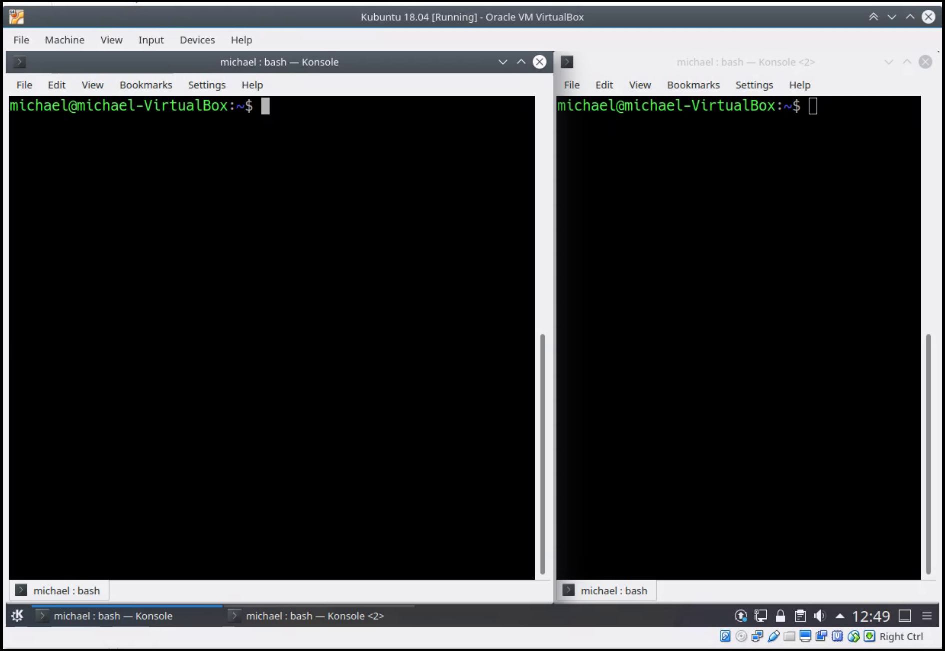


- SGX protects software from malicious environments
- Thwarts static and dynamic (= performance counters) analysis

## (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

STEALTH LEVEL: EXPERT

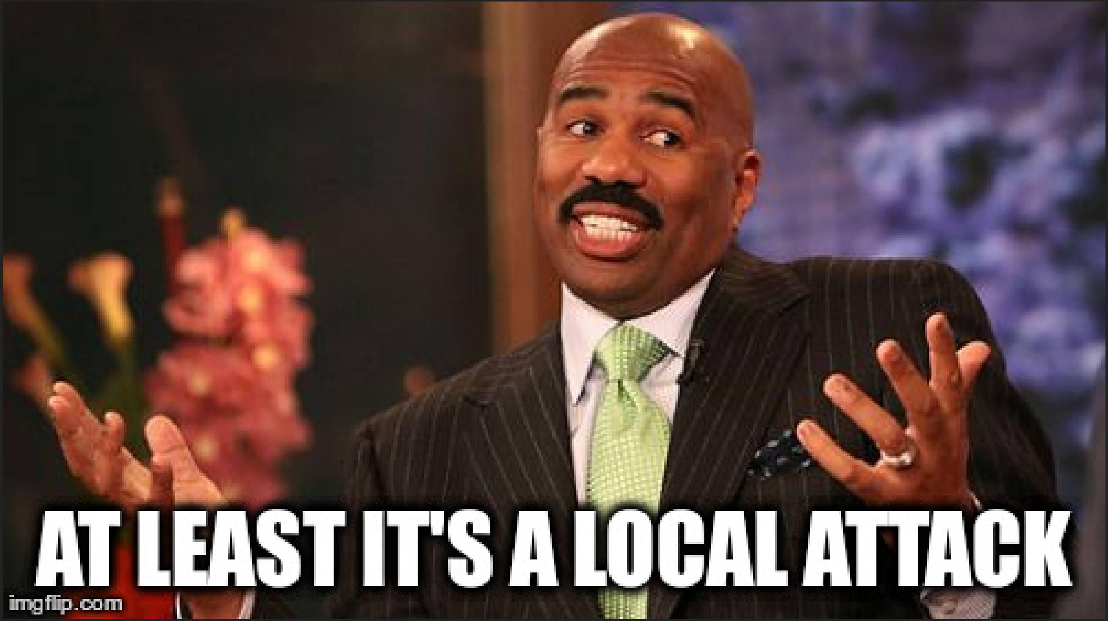

Kubuntu 18.04 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
michael : bash — Konsole
File Edit View Bookmarks Settings Help
michael : bash — Konsole <2>
michael : bash
12:49
Right Ctrl

```
michael@michael-VirtualBox:~$
```

```
michael@michael-VirtualBox:~$
```

## Bypassing the Defenses

| 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** | ● | ● | ● | ● | ● |


AT LEAST IT'S A LOCAL ATTACK
imgflip.com

## Just comparing some performance numbers...

## Just comparing some performance numbers...

## Just comparing some performance numbers...

- $\geq$ 43 000 hammering attempts (within 64 ms) for a bit flip [GMM16]

## Just comparing some performance numbers...

- $\geq$ 43 000 hammering attempts (within 64 ms) for a bit flip [GMM16]

$=$ 671 875 accesses per second

## Just comparing some performance numbers...

- $\geq$ 43 000 hammering attempts (within 64 ms) for a bit flip [GMM16]

$=$ 671 875 accesses per second

- Network packets access memory location up to 6 times (depending on kernel)

## Just comparing some performance numbers...

- $\geq$ 43 000 hammering attempts (within 64 ms) for a bit flip [GMM16]

$=$ 671 875 accesses per second

- Network packets access memory location up to 6 times (depending on kernel)

$\rightarrow$ 111 979 packets per second

## Just comparing some performance numbers...

- $\geq$ 43 000 hammering attempts (within 64 ms) for a bit flip [GMM16]

$=$ 671 875 accesses per second

- Network packets access memory location up to 6 times (depending on kernel)

$\rightarrow$ 111 979 packets per second

- Network packets are a least 64 B

## Just comparing some performance numbers...

- $\geq$ 43 000 hammering attempts (within 64 ms) for a bit flip [GMM16]

$=$ 671 875 accesses per second

- Network packets access memory location up to 6 times (depending on kernel)

$\rightarrow$ 111 979 packets per second

- Network packets are a least 64 B

$=$ 7 166 656 B/s = 7 MB/s = 57 Mb/s

## Just comparing some performance numbers...

- $\geq$ 43 000 hammering attempts (within 64 ms) for a bit flip [GMM16]

$=$ 671 875 accesses per second

- Network packets access memory location up to 6 times (depending on kernel)

$\rightarrow$ 111 979 packets per second

- Network packets are a least 64 B

$=$ 7 166 656 B/s = 7 MB/s = 57 Mb/s

$\rightarrow$ That sounds doable on “modern” networks

## Just comparing some performance numbers...

- $\geq$ 43 000 hammering attempts (within 64 ms) for a bit flip [GMM16]

$=$ 671 875 accesses per second

- Network packets access memory location up to 6 times (depending on kernel)

$\rightarrow$ 111 979 packets per second

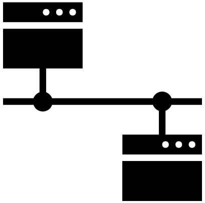
- Network packets are a least 64 B

$=$ 7 166 656 B/s = 7 MB/s = 57 Mb/s

$\rightarrow$ That sounds doable on “modern” networks

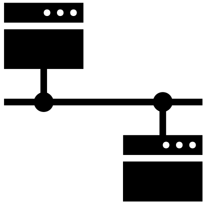
## Nethammer

## Nethammer

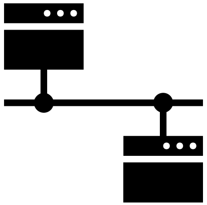


## Nethammer

Inducing bit flips:



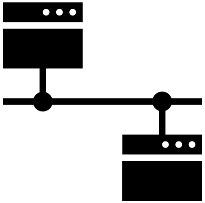
## Nethammer



Inducing bit flips:

- Network stacks on ARM often use uncached memory (perfect for hammering)

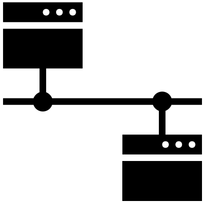
## Nethammer



Inducing bit flips:

- Network stacks on ARM often use uncached memory (perfect for hammering)
- Intel recommends Intel CAT for QoS (perfect for hammering)

## Nethammer

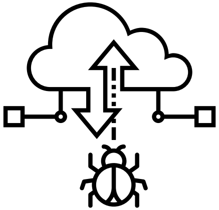


Inducing bit flips:

- Network stacks on ARM often use uncached memory (perfect for hammering)
- Intel recommends Intel CAT for QoS (perfect for hammering)
- Network reachable code might use `clflush` or non-temporal stores (both great for hammering)

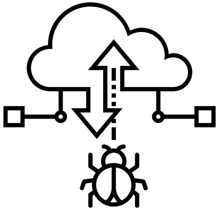
## Exploiting Nethammer Bit Flips

## Exploiting Nethammer Bit Flips



## Exploiting Nethammer Bit Flips

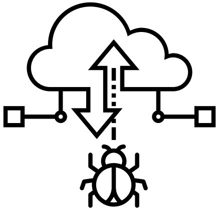
Nethammer on ...



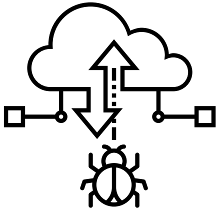
## Exploiting Nethammer Bit Flips

Nethammer on ...

- SGX = powerful DoS



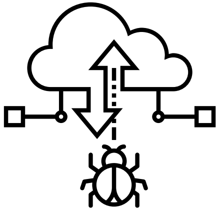
## Exploiting Nethammer Bit Flips



Nethammer on ...

- SGX = powerful DoS
- File system = persistent DoS

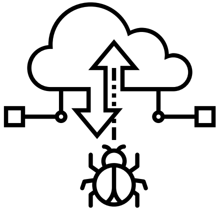
## Exploiting Nethammer Bit Flips



Nethammer on ...

- SGX = powerful DoS
- File system = persistent DoS
- DNS entries on a DNS server = bit-squatting attack

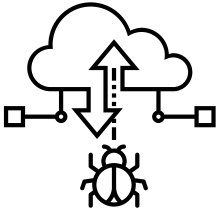
## Exploiting Nethammer Bit Flips



Nethammer on ...

- SGX = powerful DoS
- File system = persistent DoS
- DNS entries on a DNS server = bit-squatting attack
- OCSP servers = make invalid certificates great again

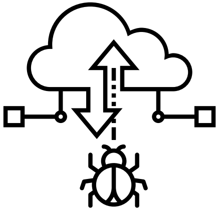
## Exploiting Nethammer Bit Flips



Nethammer on ...

- SGX = powerful DoS
- File system = persistent DoS
- DNS entries on a DNS server = bit-squatting attack
- OCSP servers = make invalid certificates great again
- Crypto = generate private keys for broken public keys

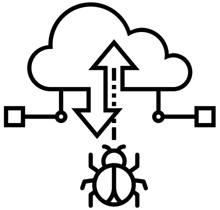
## Exploiting Nethammer Bit Flips



Nethammer on ...

- SGX = powerful DoS
- File system = persistent DoS
- DNS entries on a DNS server = bit-squatting attack
- OCSP servers = make invalid certificates great again
- Crypto = generate private keys for broken public keys
- Crypto + GitLab = manipulate repositories in the name of someone else

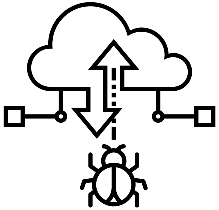
## Exploiting Nethammer Bit Flips



Nethammer on ...

- SGX = powerful DoS
- File system = persistent DoS
- DNS entries on a DNS server = bit-squatting attack
- OCSP servers = make invalid certificates great again
- Crypto = generate private keys for broken public keys
- Crypto + GitLab = manipulate repositories in the name of someone else
    - Bonus: evict the broken key and all traces are gone!

## Exploiting Nethammer Bit Flips



Nethammer on ...

- SGX = powerful DoS
- File system = persistent DoS
- DNS entries on a DNS server = bit-squatting attack
- OCSP servers = make invalid certificates great again
- Crypto = generate private keys for broken public keys
- Crypto + GitLab = manipulate repositories in the name of someone else
    - Bonus: evict the broken key and all traces are gone!
    - Original key owner will have a hard time proving that this was an attacker

## 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 [Gru+18]

## What do we learn from it?

- Many (academic) countermeasures were proposed to mitigate Rowhammer
- We showed that all of them can be circumvented [Gru+18]
- 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 [Gru+18]
- 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?

Apple had a great idea:

- Lower refresh rate → save energy + more flips

## What do we learn from it?

Apple had a great idea:

- Lower refresh rate $\rightarrow$ save energy + more flips

$\rightarrow$ ECC memory $\rightarrow$ fewer flips

## What do we learn from it?

Apple had a great idea:

- Lower refresh rate $\rightarrow$ save energy + more flips

$\rightarrow$ ECC memory $\rightarrow$ fewer flips

- It’s an optimization problem.

## What do we learn from it?

Apple had a great idea:

- Lower refresh rate $\rightarrow$ save energy + more flips

$\rightarrow$ ECC memory $\rightarrow$ fewer flips

- It's an optimization problem.
  - Too aggressive? $\rightarrow$ bit flips

## What do we learn from it?

Apple had a great idea:

- Lower refresh rate → save energy + more flips

→ ECC memory → fewer flips

- It’s an optimization problem.
  - Too aggressive? → bit flips
  - Too cautious? → waste of energy

## What do we learn from it?

Apple had a great idea:

- Lower refresh rate → save energy + more flips

→ ECC memory → fewer flips

- It's an optimization problem.
  - Too aggressive? → bit flips
  - Too cautious? → waste of energy
  - What if the “too aggressive” changes over time?

## What do we learn from it?

Apple had a great idea:

- Lower refresh rate $\rightarrow$ save energy + more flips

$\rightarrow$ ECC memory $\rightarrow$ fewer flips

- It's an optimization problem.
  - Too aggressive? $\rightarrow$ bit flips
  - Too cautious? $\rightarrow$ waste of energy
  - What if the "too aggressive" changes over time?
  - What if attackers come up with slightly better attacks?

## What do we learn from it?

Apple had a great idea:

- Lower refresh rate $\rightarrow$ save energy + more flips

$\rightarrow$ ECC memory $\rightarrow$ fewer flips

- It’s an optimization problem.
  - Too aggressive? $\rightarrow$ bit flips
  - Too cautious? $\rightarrow$ waste of energy
  - What if the “too aggressive” changes over time?
  - What if attackers come up with slightly better attacks?

  $\rightarrow$ Difficult to optimize with an adversary working against you

## 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 do we learn from it?

- We underestimated side-channel attacks for a long time

## What do 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 do we learn from it?

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

## What do we learn from it?

- We underestimated side-channel attacks for a long time
- Industry and customers have to reconsider priorities → 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

# Another Flip in the Row

Daniel Gruss, Moritz Lipp, Michael Schwarz

August 9, 2018

Graz University of Technology

## References

N. A. Anagnostopoulos, T. Arul, Y. Fan, C. Hatzfeld, A. Schaller, W. Xiong, M. Jain, M. U. Saleem, J. Lotichius, S. Gabmeyer, et al. Intrinsic Run-Time Row Hammer PUFs: Leveraging the Row Hammer Effect for Run-Time Cryptography and Improved Security. In: Cryptography 2.3 (2018), p. 13.

Z. B. Aweke, S. F. Yitbarek, R. Qiao, R. Das, M. Hicks, Y. Oren, and T. Austin. ANVIL: Software-based protection against next-generation Rowhammer attacks. In: ACM SIGPLAN Notices 51.4 (2016), pp. 743–755.

S. Bhattacharya and D. Mukhopadhyay. Curious Case of Rowhammer: Flipping Secret Exponent Bits Using Timing Analysis. In: Conference on Cryptographic Hardware and Embedded Systems (CHES). 2016.

E. Bosman, K. Razavi, H. Bos, and C. Giuffrida. Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector. In: S&P. 2016.

F. Brasser, L. Davi, D. Gens, C. Liebchen, and A.-R. Sadeghi. CAn't Touch This: Software-only Mitigation against Rowhammer Attacks targeting Kernel Memory. In: USENIX Security Symposium. 2017.

J. Corbet. Defending against Rowhammer in the kernel. Oct. 2016. URL: `https://lwn.net/Articles/704920/`.

M. Chiappetta, E. Savas, and C. Yilmaz. Real time detection of cache-based side-channel attacks using Hardware Performance Counters. Cryptology ePrint Archive, Report 2015/1034. 2015.

Y. Cheng, Z. Zhang, and S. Nepal. Still Hammerable and Exploitable: on the Effectiveness of Software-only Physical Kernel Isolation. In: arXiv:1802.07060 (2018).

H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu. Memory power management via dynamic voltage/frequency scaling. In: ACM International Conference on Autonomic Computing. 2011.

P. Frigo, C. Giuffrida, H. Bos, and K. Razavi. Grand Pwning Unit: Accelerating Microarchitectural Attacks with the GPU. In: IEEE S&P. 2018.

D. Gruss, C. Maurice, and S. Mangard. Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript. In: DIMVA. 2016.

D. Gruss, C. Maurice, K. Wagner, and S. Mangard. Flush+Flush: A Fast and Stealthy Cache Attack. In: DIMVA. 2016.

D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. O'Connell, W. Schoechl, and Y. Yarom. Another Flip in the Wall of Rowhammer Defenses. In: S&P. 2018.

N. Herath and A. Fogh. These are Not Your Grand Daddys CPU Performance Counters – CPU Hardware Performance Counters for Security. In: Black Hat Briefings. Aug. 2015. URL: https://www.blackhat.com/docs/us-15/materials/us-15-Herath-These-Are-Not-Your-Grand-Daddys-CPU-Performance-Counters-CPU-Hardware-Performance-Counters-For-Security.pdf.

G. Irazoqui, T. Eisenbarth, and B. Sunar. MASCAT: Stopping Microarchitectural Attacks Before Execution. Cryptology ePrint Archive, Report 2016/1196. 2017.

Y. Jang, J. Lee, S. Lee, and T. Kim. SGX-Bomb: Locking Down the Processor via Rowhammer Attack. In: SysTEX. 2017.

Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In: ISCA'14. 2014.

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

M. Lipp, M. T. Aga, M. Schwarz, D. Gruss, C. Maurice, L. Raab, and L. Lamster. Nethammer: Inducing Rowhammer Faults through Network Requests. In: arXiv:1711.08002 (2017).

M. Payer. HexPADS: a platform to detect "stealth" attacks. In: ESSoS'16. 2016.

P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard. DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks. In: USENIX Security Symposium. 2016.

D. Poddebniak, J. Somorovsky, S. Schinzel, M. Lochter, and P. Rösler. Attacking deterministic signature schemes using fault attacks. In: EuroS&P. 2018.

R. Qiao and M. Seaborn. A New Approach for Rowhammer Attacks. In: International Symposium on Hardware Oriented Security and Trust. 2016.

K. Razavi, B. Gras, E. Bosman, B. Preneel, C. Giuffrida, and H. Bos. Flip Feng Shui: Hammering a Needle in the Software Stack. In: USENIX Security Symposium. 2016.

M. Seaborn and T. Dullien. Exploiting the DRAM rowhammer bug to gain kernel privileges. In: Black Hat Briefings. 2015.

A. Tatar, R. Krishnan, E. Athanasopoulos, C. Giuffrida, H. Bos, and K. Razavi. Throwhammer: Rowhammer Attacks over the Network and Defenses. In: USENIX ATC. 2018.

V. van der Veen, Y. Fratantonio, M. Lindorfer, D. Gruss, C. Maurice, G. Vigna, H. Bos, K. Razavi, and C. Giuffrida. Drammer: Deterministic Rowhammer Attacks on Mobile Platforms. In: CCS'16. 2016.

V. van der Veen, M. Lindorfer, Y. Fratantonio, H. P. Pillai, G. Vigna, C. Kruegel, H. Bos, and K. Razavi. GuardION: Practical Mitigation of DMA-Based Rowhammer Attacks on ARM. In: DIMVA. 2018.

Y. Xiao, X. Zhang, Y. Zhang, and R. Teodorescu. One Bit Flips, One Cloud Flops: Cross-VM Row Hammer Attacks and Privilege Escalation. In: USENIX Security Symposium. 2016.

## Runtime of the attack

| Method | Bit flips | Templating | Waylaying | Total |
|---:|---:|---:|---:|---:|
| Double-sided, waylaying | 91 | 26.1 h | 69.4 h | 95.5 h |
| Single-sided, waylaying | 87 | 27.5 h | 70.6 h | 98.1 h |
| One-location, waylaying | 50 | 47.3 h | 90.5 h | 137.8 h |
| Double-sided, chasing | 1 | 0.7 h | 43.7 h | 44.4 h |
| Single-sided, chasing | 1 | 0.7 h | 43.7 h | 44.4 h |
| One-location, chasing | 1 | 1.3 h | 44.0 h | 45.4 h |