TRRespass: Exploiting the many sides of Target Row Refresh

Pietro Frigo, Emanuele Vannacci, Hasan Hassan, Victor van der Veen, Onur Mutlu, Cristiano Giuffrida, Herbert Bos, Kaveh Razavi

2020 IEEE Symposium on Security and Privacy
18-21 May 2020

Julian Müller

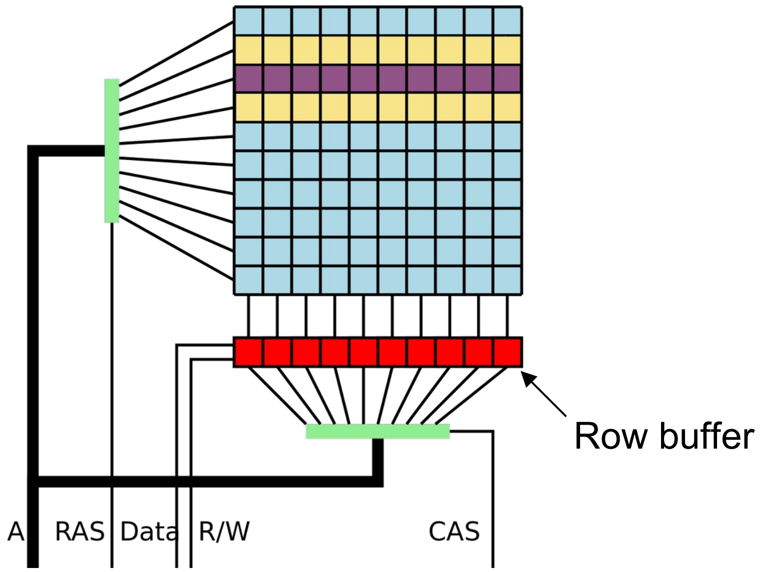
Motivation

- RowHammer discovered in 2014
- Used as attack vector in the wild
- Aim of the paper: Analyze and circumvent mitigation mechanisms

Overview

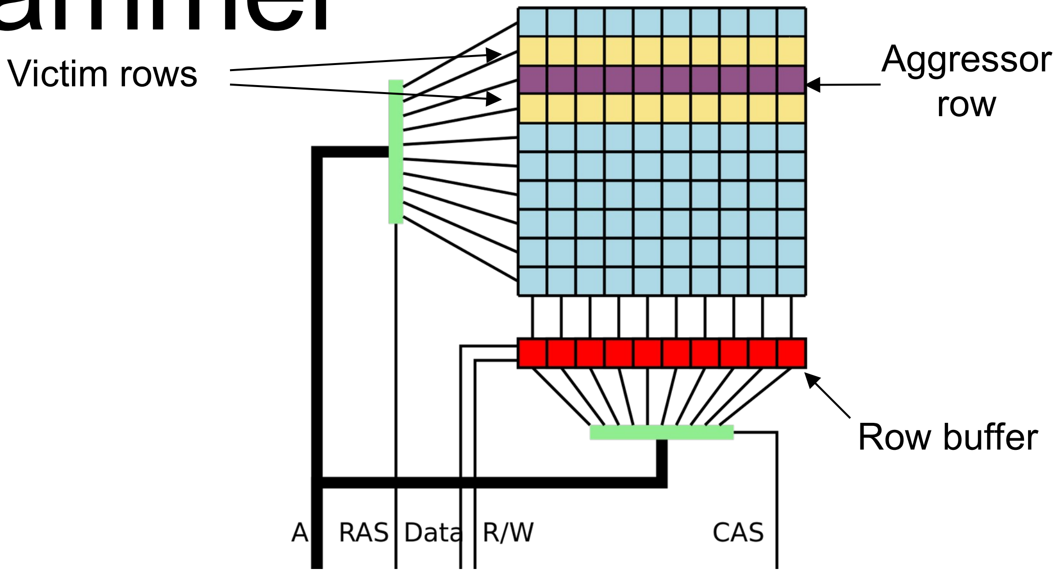
DRAM

- Organized into rows and columns
- Memory cells leak => refresh every 64ms
- Content of a row is loaded into row buffer
- Electromagnetic field produced by row activation increases leakage => bit may flip



RowHammer

- Electromagnetic field produced by row activation increases leakage => bit may flip
- Alternately read addresses X and Y
- Flush the cache
- Clflush is an unprivileged instruction on x86



```
code1a:
mov (X), %eax // read from address X
mov (Y), %ebx // read from address Y
clflush (X) // flush cache for address X
clflush (Y) // flush cache for address Y
jmp code1a
```

Exploiting RowHammer

- Change contents of read-only memory, e.g. shared libaries
- Change contents of virtual pages of other processes/kernel

- Without the OS noticing it

RowHammer breaks memory isolation!!!

RowHammer based attacks

- Manipulate Page Tables to gain access to the whole physical Memory[1]
- Breaking out of NaCl Sandbox[1]
- Emulate clflush in a web browser using JavaScript[2]
- Read out secred data, e.g. an RSA-Key[3]

1. *M. Seaborn and T. Dullien, "Exploiting the DRAM Rowhammer Bugto Gain Kernel Privileges," Black Hat USA, 2015*
2. *D. Grusset al., "Rowhammer.js: A Remote Software-Induced FaultAttack in JavaScript," DIMVA, 2016*
3. *A. Kwong et al., "RAMBleed: Reading Bits in Memory WithoutAccessing Them," inS&P, 2020*

Mitigation (so far)

- Doubling (or even quadrupling) the refresh rate
  - Only solution for existing circuits
  - Energy consumption is proportional to refresh rate
  - Latency increases
  - Time frame still too big
  - =>**Ineffective**

Mitigation (so far)

- ECC memory
  - Can correct 1 bit flip per 64-bit word
  - Can detect 2 bit flips per 64-bit word
  - Cannot detect 3 or more bit flips
  - RowHammer usually induces more bit flips
  - =>**Ineffective**

Mitigation (so far)

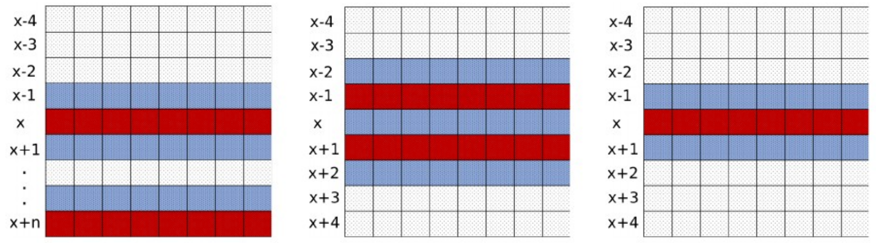
- Doubling (or even quadrupling) the refresh rate
  - =>**Ineffective**
- ECC memory
  - =>**Ineffective**
- Target Row Refresh (TRR)
  - Count accesses per row
  - Issue extra refreshes to victim rows
- DRAM manufacturers advertise their chips as RowHammer-free
- DRAM manufacturers do not disclose their TRR implementations

Industry on RowHammer:

"We have it totally under control"
"One day, it's like a miracle, it will disappear"
"It will go away, and we're going to have a great victory"
"It is going away"

Hammering

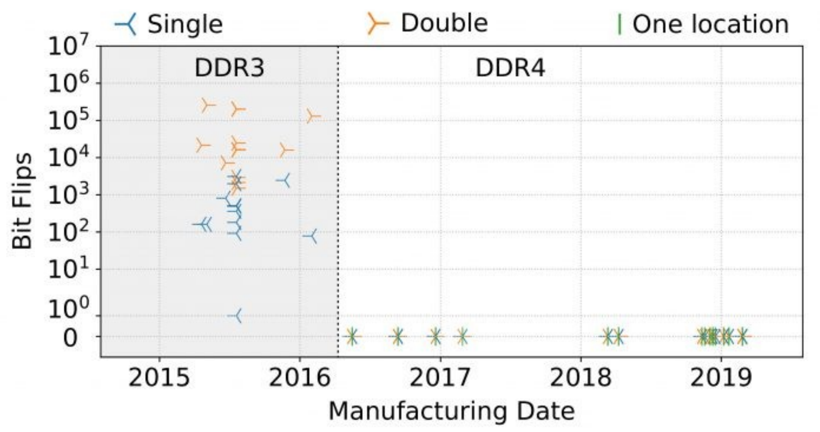
- Single sided: Standard pattern used in original demonstration
- Double sided: victim row in the middle experiences twice as many hammerings



(a) Single-sided (b) Double-sided (c) One-location

Hammering DDR4

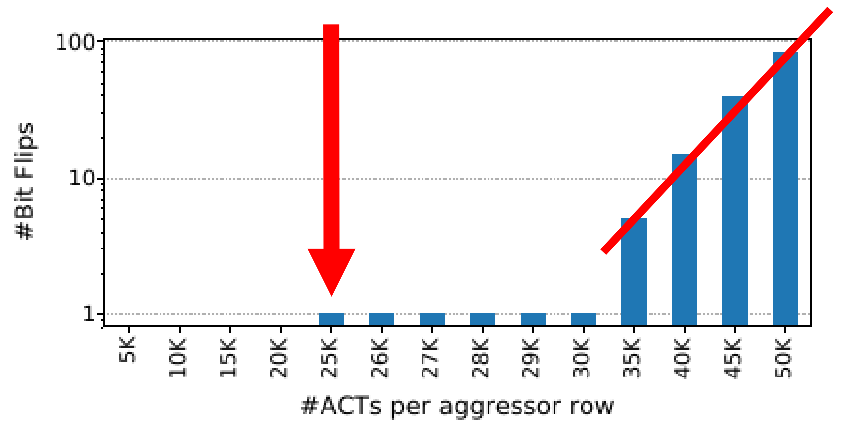
- 2016: 87% of all modules vulnerable (DDR3)
- Analysis of 42 recent modules (DDR4) (Samsung, Micron, Hynix)
- Standard hammering patterns
- No bit flips observable



Hammering DDR4

- Refreshing turned off
- Double sided hammering
- Bit flips at 25K activations per row
- 139K needed on DDR3
- Exponential growth

C_{12}:

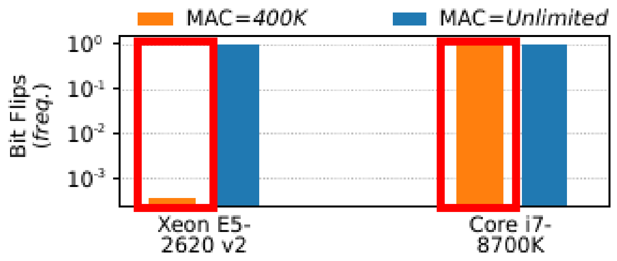


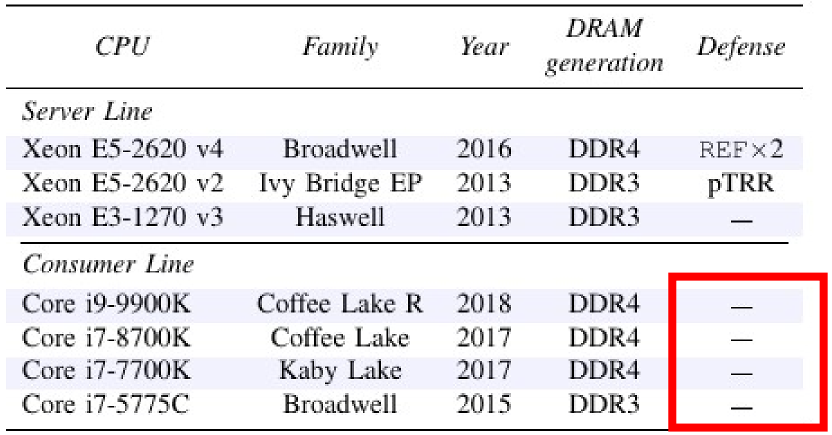
Intel‘s pTRR

- Only publicly advertised MC-implementation of TRR
- MAC-value inside DRAM chips:
  - any positive number => issue refresh if that number is reached
  - Untested => double refresh rate
  - Unlimited => do nothing

Intel‘s pTRR

- MAC=400k vs. MAC=Unlimited
- Core i7 vs. Xeon E5
- Xeon E5: almost no bit flips with MAC=400k
- Core i7: No difference
- No pTRR on consumer line CPUs





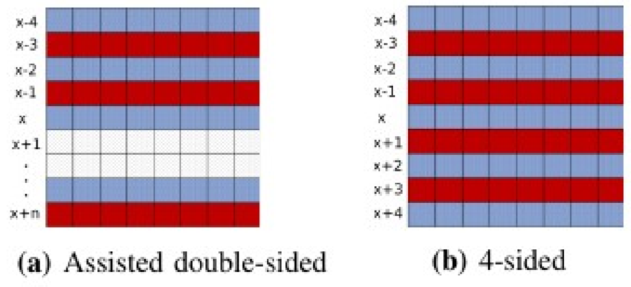
CPU	*Family*	*Year*	*DRAM generation*	*Defense*
Server Line				
Xeon E5-2620 v4	Broadwell	2016	DDR4	REF×2
Xeon E5-2620 v2	Ivy Bridge EP	2013	DDR3	pTRR
Xeon E3-1270 v3	Haswell	2013	DDR3	—
Consumer Line				
Core i9-9900K	Coffee Lake R	2018	DDR4	—
Core i7-8700K	Coffee Lake	2017	DDR4	—
Core i7-7700K	Kaby Lake	2017	DDR4	—
Core i7-5775C	Broadwell	2015	DDR3	—

Reverse-engineering in-DRAM TRR

- Hypothesis:
  - Sampler: detects potential aggressor rows
  - Inhibitor: issues additional refreshes to victim rows

TRRespass

- New Version of RowHammer code
- Extends double-sided to n-sided access pattern



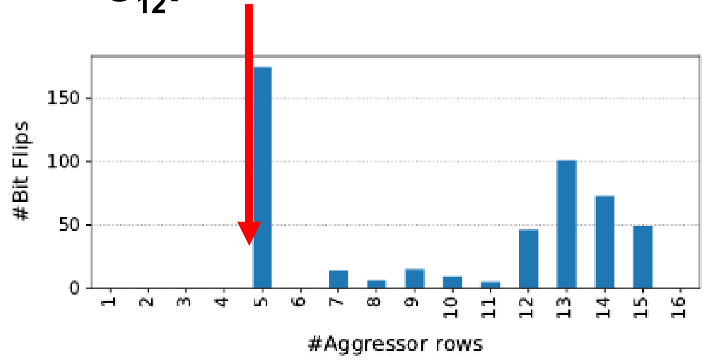
(a) Assisted double-sided (b) 4-sided

#MakeDoubleSidedGreatAgain

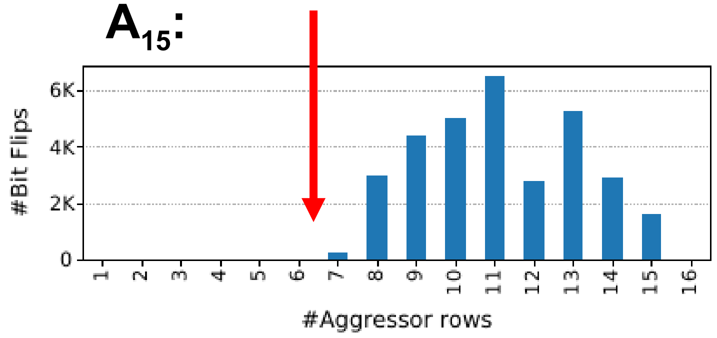
Determining the size of the sampler

- Increasing n reveals sampler size
  - 4 on C_{12}
  - 6 on A_{15}
- Exploitable
  - Use n dummy aggressor rows
  - Camouflage real aggressor rows

C_{12}:



A_{15}:



TRRespass

TRRespass: The full evaluation

- TRRespass running for 6 hours
- 13 of 42 modules vulnerable (31%)
- 87% after discovery
- Large divergence in # of bit flips
  - 5 bit flips on B_2
  - 190k bit flips on C_{12}
- Only 2 of C's modules vulnerable

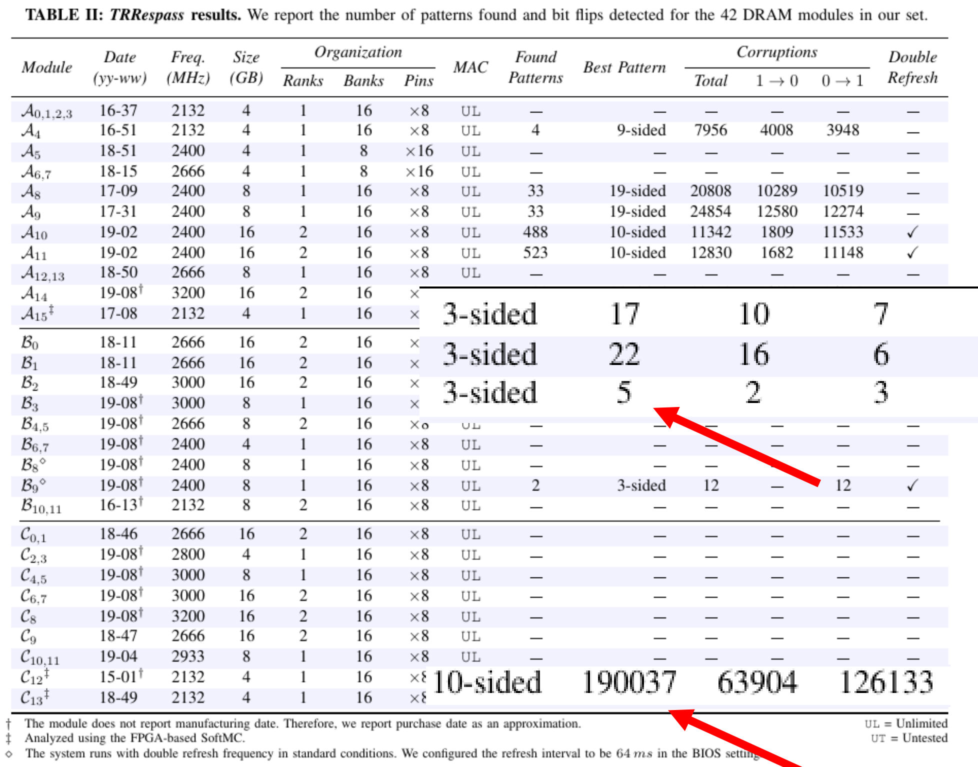


TABLE II: *TRRespass* results. We report the number of patterns found and bit flips detected for the 42 DRAM modules in our set.

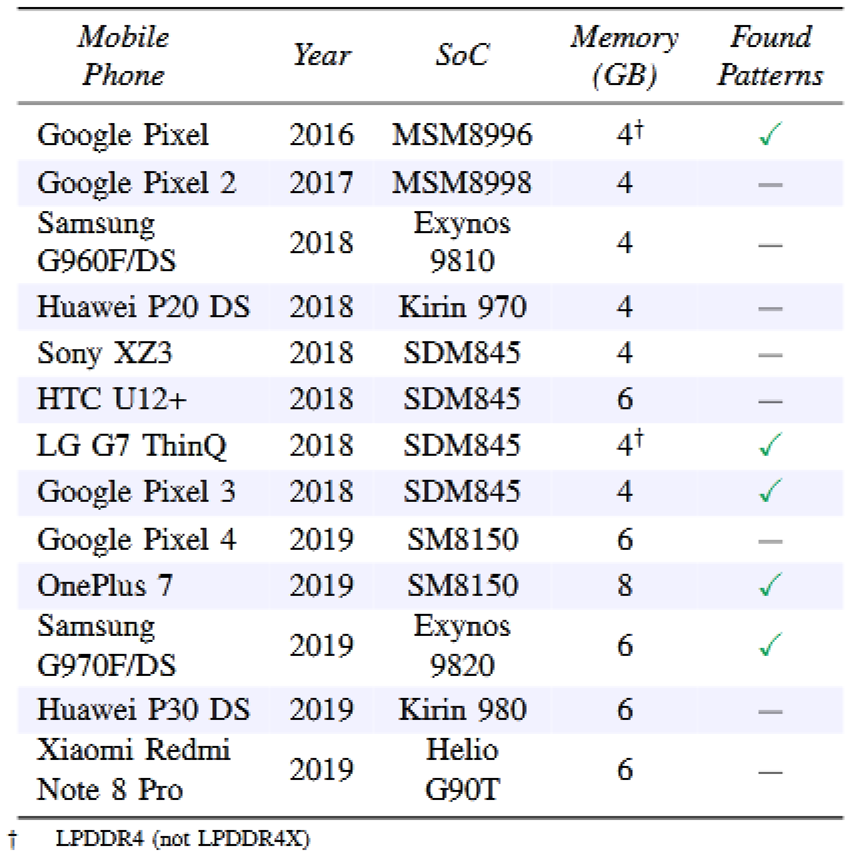
Module	Date (yy-ww)	Freq. (MHz)	Size (GB)	Organization			MAC	Found Patterns	Best Pattern	Corruptions			Double Refresh
				Ranks	Banks	Pins				Total	$1 \to 0$	$0 \to 1$	
$\mathcal{A}_{0,1,2,3}$	16-37	2132	4	1	16	×8	UL	—	—	—	—	—	—
$\mathcal{A}_4$	16-51	2132	4	1	16	×8	UL	4	9-sided	7956	4008	3948	—
$\mathcal{A}_5$	18-51	2400	4	1	8	×16	UL	—	—	—	—	—	—
$\mathcal{A}_{6,7}$	18-15	2666	4	1	8	×16	UL	—	—	—	—	—	—
$\mathcal{A}_8$	17-09	2400	8	1	16	×8	UL	33	19-sided	20808	10289	10519	—
$\mathcal{A}_9$	17-31	2400	8	1	16	×8	UL	33	19-sided	24854	12580	12274	—
$\mathcal{A}_{10}$	19-02	2400	16	2	16	×8	UL	488	10-sided	11342	1809	11533	✓
$\mathcal{A}_{11}$	19-02	2400	16	2	16	×8	UL	523	10-sided	12830	1682	11148	✓
$\mathcal{A}_{12,13}$	18-50	2666	8	1	16	×8	UL	—	—	—	—	—	—
$\mathcal{A}_{14}$	19-08†	3200	16	2	16	×							
$\mathcal{A}_{15}$‡	17-08	2132	4	1	16	×							
$\mathcal{B}_0$	18-11	2666	16	2	16	×			3-sided	17	10	7	
$\mathcal{B}_1$	18-11	2666	16	2	16	×			3-sided	22	16	6	
$\mathcal{B}_2$	18-49	3000	16	2	16	×			3-sided	5	2	3	
$\mathcal{B}_3$	19-08†	3000	8	1	16	×							
$\mathcal{B}_{4,5}$	19-08†	2666	8	2	16	×8	UL	—	—	—	—	—	—
$\mathcal{B}_{6,7}$	19-08†	2400	4	1	16	×8	UL	—	—	—	—	—	—
$\mathcal{B}_8$◇	19-08†	2400	8	1	16	×8	UL	—	—	—	—	—	—
$\mathcal{B}_9$◇	19-08†	2400	8	1	16	×8	UL	2	3-sided	12	—	12	✓
$\mathcal{B}_{10,11}$	16-13†	2132	8	2	16	×8	UL	—	—	—	—	—	—
$\mathcal{C}_{0,1}$	18-46	2666	16	2	16	×8	UL	—	—	—	—	—	—
$\mathcal{C}_{2,3}$	19-08†	2800	4	1	16	×8	UL	—	—	—	—	—	—
$\mathcal{C}_{4,5}$	19-08†	3000	8	1	16	×8	UL	—	—	—	—	—	—
$\mathcal{C}_{6,7}$	19-08†	3000	16	2	16	×8	UL	—	—	—	—	—	—
$\mathcal{C}_8$	19-08†	3200	16	2	16	×8	UL	—	—	—	—	—	—
$\mathcal{C}_9$	18-47	2666	16	2	16	×8	UL	—	—	—	—	—	—
$\mathcal{C}_{10,11}$	19-04	2933	8	1	16	×8	UL	—	—	—	—	—	—
$\mathcal{C}_{12}$‡	15-01†	2132	4	1	16	×8			10-sided	190037	63904	126133	
$\mathcal{C}_{13}$‡	18-49	2132	4	1	16	×8							

† The module does not report manufacturing date. Therefore, we report purchase date as an approximation.
‡ Analyzed using the FPGA-based SoftMC.
◇ The system runs with double refresh frequency in standard conditions. We configured the refresh interval to be 64 ms in the BIOS setting

UL = Unlimited
UT = Untested

TRRespass on phones

- 13 models tested
  - Only Android
- 5 of 13 vulnerable (38%)
- Different DRAM chips accross the same model



Mobile Phone	*Year*	*SoC*	*Memory (GB)*	*Found Patterns*
Google Pixel	2016	MSM8996	4†	✓
Google Pixel 2	2017	MSM8998	4	—
Samsung G960F/DS	2018	Exynos 9810	4	—
Huawei P20 DS	2018	Kirin 970	4	—
Sony XZ3	2018	SDM845	4	—
HTC U12+	2018	SDM845	6	—
LG G7 ThinQ	2018	SDM845	4†	✓
Google Pixel 3	2018	SDM845	4	✓
Google Pixel 4	2019	SM8150	6	—
OnePlus 7	2019	SM8150	8	✓
Samsung G970F/DS	2019	Exynos 9820	6	✓
Huawei P30 DS	2019	Kirin 980	6	—
Xiaomi Redmi Note 8 Pro	2019	Helio G90T	6	—

† LPDDR4 (not LPDDR4X)

Mounting real-world attacks using TRRespass

- Attacks tested:
  - Manipulating page tables
  - Corrupting RSA-Key
  - Circumventing sudo checks
- Most and least vulnerable modules from each vendor
- All attacks failed on B‘s modules
- Time span between 2s and 3h

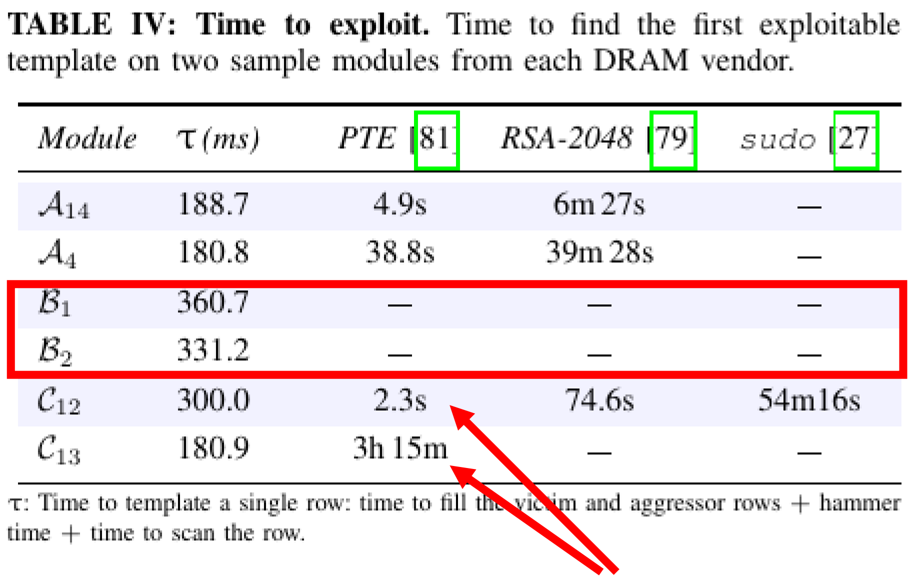


TABLE IV: Time to exploit. Time to find the first exploitable template on two sample modules from each DRAM vendor.

Module	τ *(ms)*	*PTE* [81]	*RSA-2048* [79]	sudo [27]
$\mathcal{A}_{14}$	188.7	4.9s	6m 27s	–
$\mathcal{A}_{4}$	180.8	38.8s	39m 28s	–
$\mathcal{B}_{1}$	360.7	–	–	–
$\mathcal{B}_{2}$	331.2	–	–	–
$\mathcal{C}_{12}$	300.0	2.3s	74.6s	54m16s
$\mathcal{C}_{13}$	180.9	3h 15m	–	–

τ: Time to template a single row: time to fill the victim and aggressor rows + hammer time + time to scan the row.

Conclusion

- RowHammer is still a problem
- Partially worse
  - 2016: 139k activations for bit flips
  - 2020: 50k
- Partially better
  - 2016: 87% of all modules vulnerable
  - 2020: 31%

Industry:

Researchers:

Strengths

- Addressing a serious security issue
- Very detailed analysis and reverse engineering of mostly undocumented hardware
- Proving manufacturers wrong

Weaknesses

- No statement about what DRAM manufacturers did wrong
- No improvement suggestions
- TRRespass vs. pTRR??
- TRRespass vs. iPhone??
- Real-world attacks on phones??
- No disclosure of the manufacturers

Discussion

- Is such detailed public disclosure of vulnerabilities a good idea?
- Is TRR the right way to go?
- Will RowHammer kill DRAM?