Software side-channel attacks and fault attacks on ARM devices

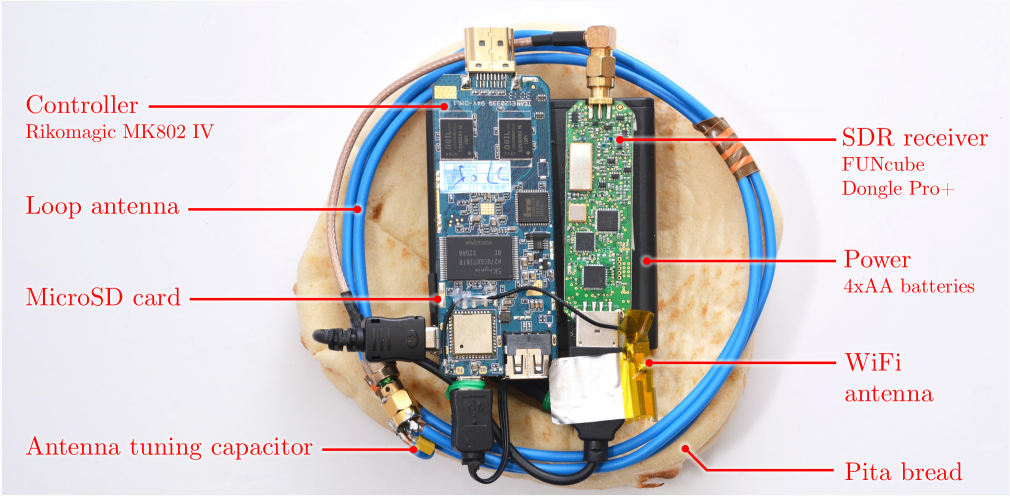
Clémentine Maurice, CNRS, IRISA, EMSEC

February 18, 2020–SILM seminar, Rennes, France

Software-based side channels and fault attacks

- power consumption, electromagnetic leaks, lasers, glitching…

Software-based side channels and fault attacks



Software-based side channels and fault attacks

- power consumption, electromagnetic leaks, lasers, glitching…
  - → targeted attacks, physical access
  - → mostly performed on embedded devices

Software-based side channels and fault attacks

- power consumption, electromagnetic leaks, lasers, glitching…
  - → targeted attacks, physical access
  - → mostly performed on embedded devices
- timing attacks, microarchitectural attacks, software-based fault attacks…

Software-based side channels and fault attacks

- power consumption, electromagnetic leaks, lasers, glitching…
  - → targeted attacks, physical access
  - → mostly performed on embedded devices
- timing attacks, microarchitectural attacks, software-based fault attacks…
  - → no physical access required
  - → require code co-location

Side-channel attacks

Cache attacks

- cache attacks $\rightarrow$ exploit timing differences of memory accesses

Cache attacks

- cache attacks $\rightarrow$ exploit timing differences of memory accesses
- attacker monitors which lines are accessed, not the content

Cache attacks

- cache attacks $\rightarrow$ exploit timing differences of memory accesses
- attacker monitors which lines are accessed, not the content
- covert channel: two processes communicating with each other
  - not allowed to do so

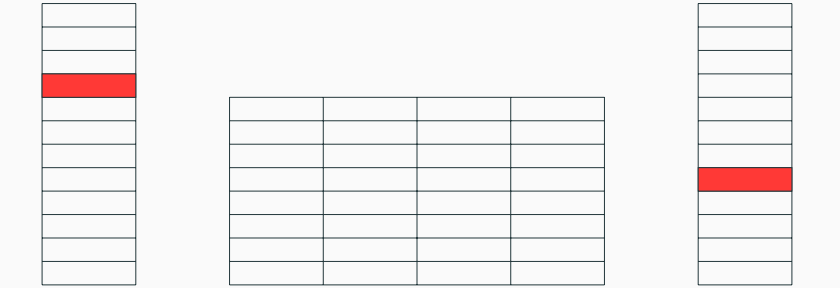
Cache attacks

- cache attacks $\rightarrow$ exploit timing differences of memory accesses
- attacker monitors which lines are accessed, not the content
- covert channel: two processes communicating with each other
  - not allowed to do so
- side-channel attack: one malicious process spies on benign processes
  - e.g., steals crypto keys, spies on keystrokes

Cache attacks

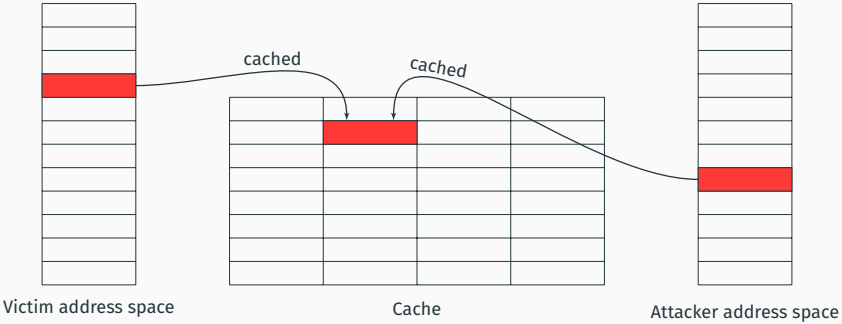
- cache attacks $\rightarrow$ exploit timing differences of memory accesses
- attacker monitors which lines are accessed, not the content
- covert channel: two processes communicating with each other
  - not allowed to do so
- side-channel attack: one malicious process spies on benign processes
  - e.g., steals crypto keys, spies on keystrokes
- different techniques can be used for both covert channels and side-channel attacks

Cache attack: Flush+Reload



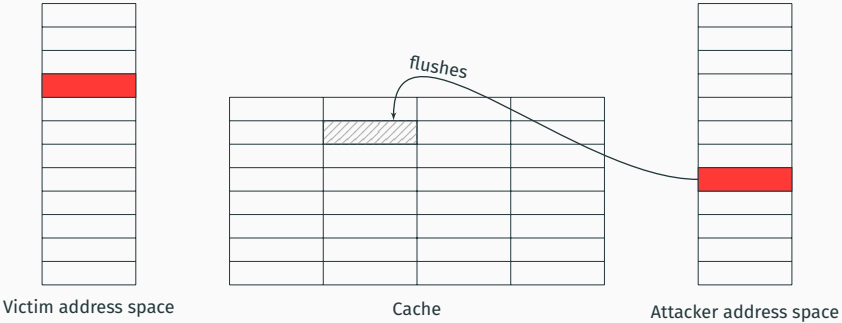
Step 1: Attacker maps shared library (shared memory, in cache)

Cache attack: Flush+Reload



Step 1: Attacker maps shared library (shared memory, in cache)

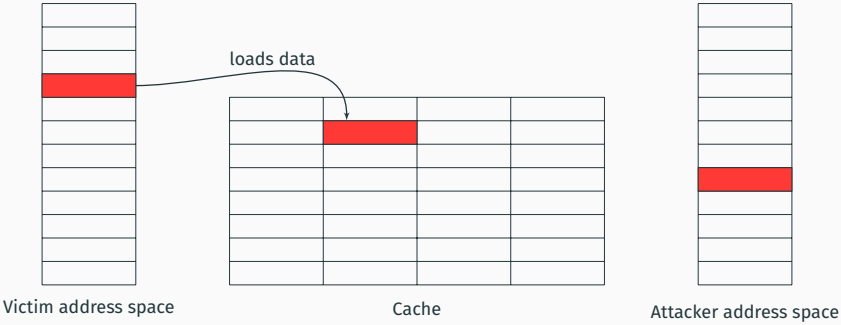
Cache attack: Flush+Reload



Step 1: Attacker maps shared library (shared memory, in cache)

Step 2: Attacker flushes the shared cache line

Cache attack: Flush+Reload

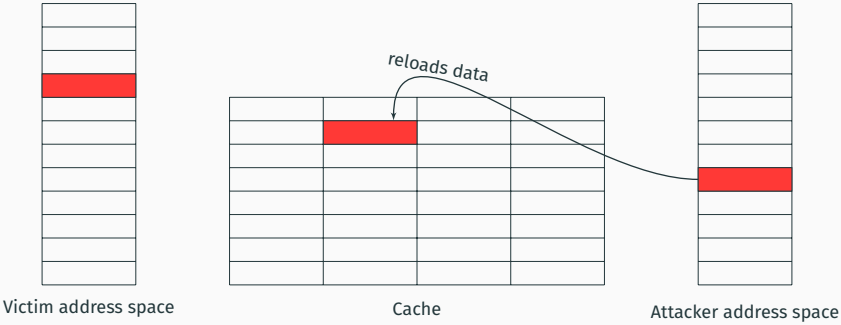


Step 1: Attacker maps shared library (shared memory, in cache)

Step 2: Attacker flushes the shared cache line

Step 3: Victim loads the data

Cache attack: Flush+Reload



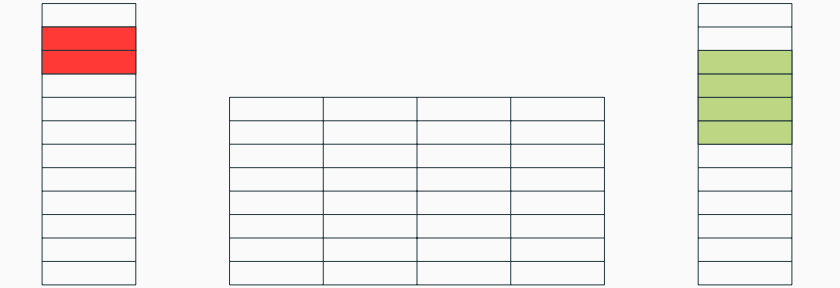
Step 1: Attacker maps shared library (shared memory, in cache)

Step 2: Attacker flushes the shared cache line

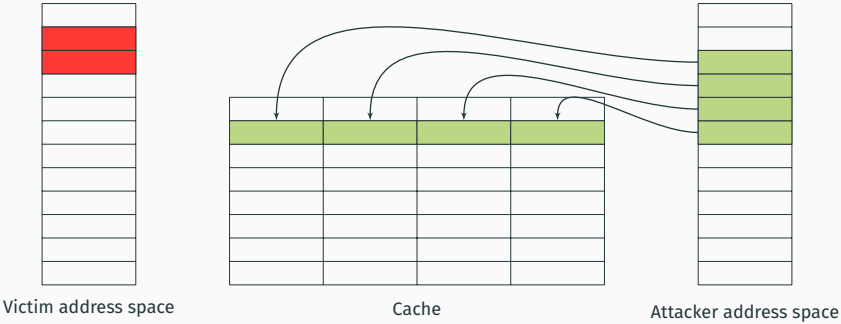
Step 3: Victim loads the data

Step 4: Attacker reloads the data

Cache attacks: Prime+Probe

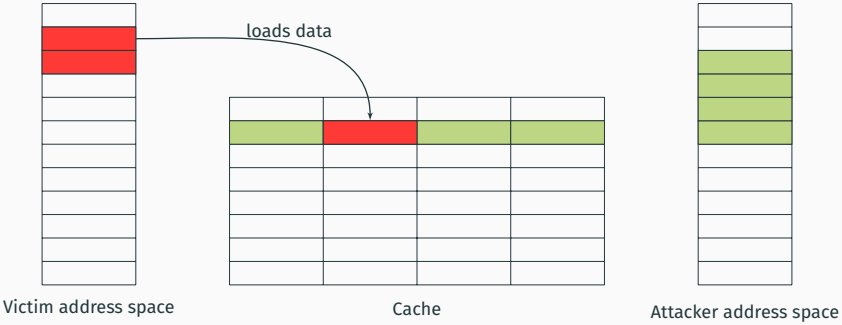


Cache attacks: Prime+Probe



Step 1: Attacker primes, *i.e.*, fills, the cache (no shared memory)

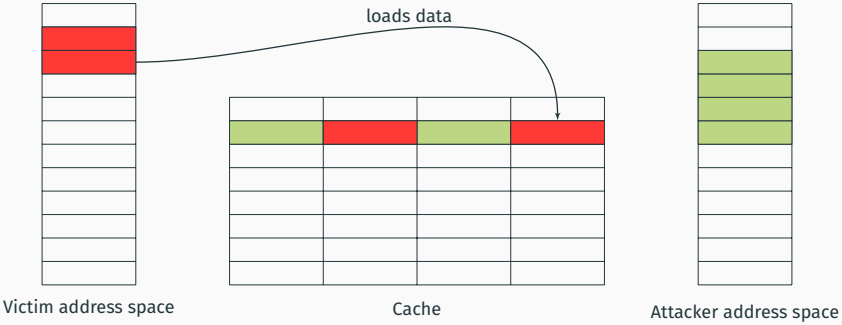
Cache attacks: Prime+Probe



Step 1: Attacker primes, *i.e.*, fills, the cache (no shared memory)

Step 2: Victim evicts cache lines while running

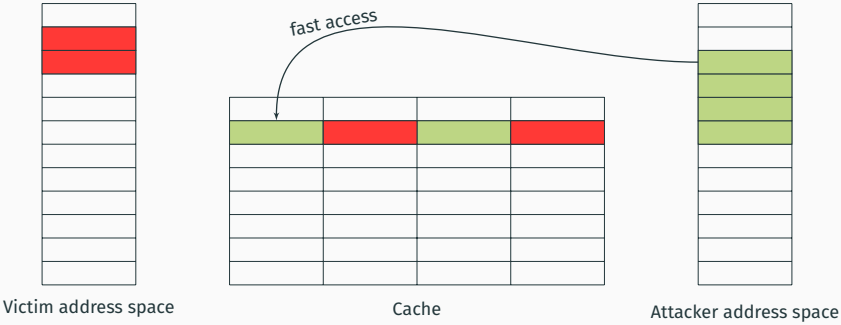
Cache attacks: Prime+Probe



Step 1: Attacker primes, *i.e.*, fills, the cache (no shared memory)

Step 2: Victim evicts cache lines while running

Cache attacks: Prime+Probe

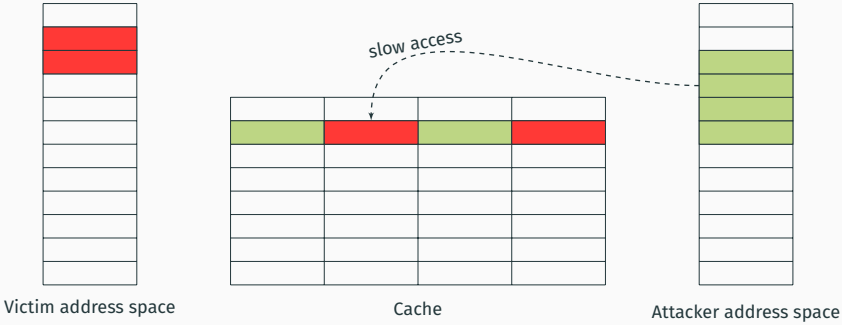


Step 1: Attacker primes, *i.e.*, fills, the cache (no shared memory)

Step 2: Victim evicts cache lines while running

Step 3: Attacker probes data to determine if set has been accessed

Cache attacks: Prime+Probe



Step 1: Attacker primes, *i.e.*, fills, the cache (no shared memory)

Step 2: Victim evicts cache lines while running

Step 3: Attacker probes data to determine if set has been accessed

A bit of history

- 2005: first cache attacks on Intel
- 2013: first cache attack on Android (ARM)
  - $\rightarrow$ attack requires privileges
- 2016: first unprivileged cache attack on ARM

Even today most attacks target Intel

C. Percival. “Cache missing for fun and profit”. In: *Proceedings of BSDCan*. 2005.

R. Spreitzer et al. “On the Applicability of Time-Driven Cache Attacks on Mobile Devices”. In: *NSS*. 2013.

M. Lipp et al. “ARMageddon: Cache Attacks on Mobile Devices”. In: *USENIX Security Symposium*. 2016.

Intel vs ARM

Challenge #1 no flush instruction on ARMv7-A
$\rightarrow$ use cache eviction

M. Lipp et al. “ARMageddon: Cache Attacks on Mobile Devices”. In: *USENIX Security Symposium*. 2016.

Intel vs ARM

Challenge #1 no flush instruction on ARMv7-A

$\rightarrow$ use cache eviction

Challenge #2 pseudo-random replacement policy

$\rightarrow$ eviction strategies that use multiple accesses

M. Lipp et al. “ARMageddon: Cache Attacks on Mobile Devices”. In: *USENIX Security Symposium*. 2016.

Intel vs ARM

Challenge #1 no flush instruction on ARMv7-A
→ use cache eviction

Challenge #2 pseudo-random replacement policy
→ eviction strategies that use multiple accesses

Challenge #3 privileged cycle counter
→ thread counter (nano-second resolution)

M. Lipp et al. “ARMageddon: Cache Attacks on Mobile Devices”. In: *USENIX Security Symposium*. 2016.

Intel vs ARM

Challenge #1 no flush instruction on ARMv7-A
→ use cache eviction

Challenge #2 pseudo-random replacement policy
→ eviction strategies that use multiple accesses

Challenge #3 privileged cycle counter
→ thread counter (nano-second resolution)

Challenge #4 non-inclusive caches
→ abuse cache coherency protocol for shared memory

M. Lipp et al. “ARMageddon: Cache Attacks on Mobile Devices”. In: *USENIX Security Symposium*. 2016.

Intel vs ARM

Challenge #1 no flush instruction on ARMv7-A
→ use cache eviction

Challenge #2 pseudo-random replacement policy
→ eviction strategies that use multiple accesses

Challenge #3 privileged cycle counter
→ thread counter (nano-second resolution)

Challenge #4 non-inclusive caches
→ abuse cache coherency protocol for shared memory

Challenge #5 no shared cache between multiple CPUs (big.LITTLE)
→ abuse cache coherency protocol for shared memory

M. Lipp et al. “ARMageddon: Cache Attacks on Mobile Devices”. In: *USENIX Security Symposium*. 2016.

Threat model

- people tend to use their browser for everything on personal computers

https://techcrunch.com/2017/05/04/report-smartphone-owners-are-using-9-apps-per-day-30-per-month/

Threat model

- people tend to use their browser for everything on personal computers
- people tend to install a new app for everything on mobile devices

https://techcrunch.com/2017/05/04/report-smartphone-owners-are-using-9-apps-per-day-30-per-month/

Threat model

- people tend to use their browser for everything on personal computers
- people tend to install a new app for everything on mobile devices
- apps on mobile devices have specific permissions

https://techcrunch.com/2017/05/04/report-smartphone-owners-are-using-9-apps-per-day-30-per-month/

Threat model

- people tend to use their browser for everything on personal computers
- people tend to install a new app for everything on mobile devices
- apps on mobile devices have specific permissions

→ covert channels make a lot of sense in this context!

https://techcrunch.com/2017/05/04/report-smartphone-owners-are-using-9-apps-per-day-30-per-month/

Threat model

- people tend to use their browser for everything on personal computers
- people tend to install a new app for everything on mobile devices
- apps on mobile devices have specific permissions

→ covert channels make a lot of sense in this context!

→ e.g., one app transmitting covertly contact information to another app that does not have the permission

https://techcrunch.com/2017/05/04/report-smartphone-owners-are-using-9-apps-per-day-30-per-month/

Covert channels

- Flush+Reload, Evict+Reload and Flush+Flush, cross-core and cross-CPU
- faster than non-microarchitectural techniques

Work	Type	Bandwidth [bps]	Error rate
Schlegel et al.	Vibration settings	87	–
Schlegel et al.	Volume settings	150	–
Schlegel et al.	File locks	685	–
Marforio et al.	UNIX socket discovery	2 600	–
Marforio et al.	Type of Intents	4 300	–
Ours (OnePlus One)	Evict+Reload, cross-core	12 537	5.00%
Ours (Alcatel One Touch Pop 2)	Evict+Reload, cross-core	13 618	3.79%
Ours (Samsung Galaxy S6)	Flush+Flush, cross-core	178 292	0.48%
Ours (Samsung Galaxy S6)	Flush+Reload, cross-CPU	257 509	1.83%
Ours (Samsung Galaxy S6)	Flush+Reload, cross-core	1 140 650	1.10%

A note on Meltdown

Vendor \ Attack	MD-US [56]	MD-P [85, 90]	MD-GP [8, 35]	MD-NM [78]	MD-RW [48]	MD-PK	MD-BR	MD-DE	MD-AC	MD-UD	MD-SS	MD-XD	MD-SM
Intel	●	●	●	●	●	★	★	☆	☆	☆	☆	☆	☆
ARM	●	○	●	_	●	_	_	☆	☆	☆	_	☆	☆
AMD	○	○	○	○	○	_	★	☆	☆	☆	☆	☆	☆

Symbols indicate whether at least one CPU model is vulnerable (filled) vs. no CPU is known to be vulnerable (empty). Glossary: reproduced (● vs. ○), first shown in this paper (★ vs. ☆), not applicable (_). All tests performed without defenses enabled.

C. Canella et al. “A Systematic Evaluation of Transient Execution Attacks and Defenses”. In: *USENIX Security*. 2019.

A note on Spectre

Method		Attack	Spectre-PHT	Spectre-BTB	Spectre-RSB	Spectre-STL
Intel	intra-process	in-place	● [48, 50]	★	● [59]	● [29]
		out-of-place	★	● [13]	● [52, 59]	○
	cross-process	in-place	★	● [13, 50]	● [52, 59]	○
		out-of-place	★	● [50]	● [52]	○
ARM	intra-process	in-place	● [48, 50]	★	● [6]	● [6]
		out-of-place	★	☆	● [6]	○
	cross-process	in-place	★	● [6, 50]	☆	○
		out-of-place	★	☆	☆	○
AMD	intra-process	in-place	● [50]	★	★	● [29]
		out-of-place	★	☆	★	○
	cross-process	in-place	★	● [50]	★	○
		out-of-place	★	☆	★	○

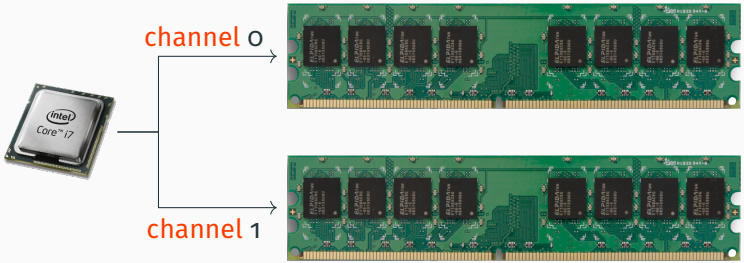
Symbols indicate whether an attack is possible and known (●), not possible and known (○), possible and previously unknown or not shown (★), or tested and did not work and previously unknown or not shown (☆). All tests performed with no defenses enabled.

C. Canella et al. “A Systematic Evaluation of Transient Execution Attacks and Defenses”. In: *USENIX Security*. 2019.

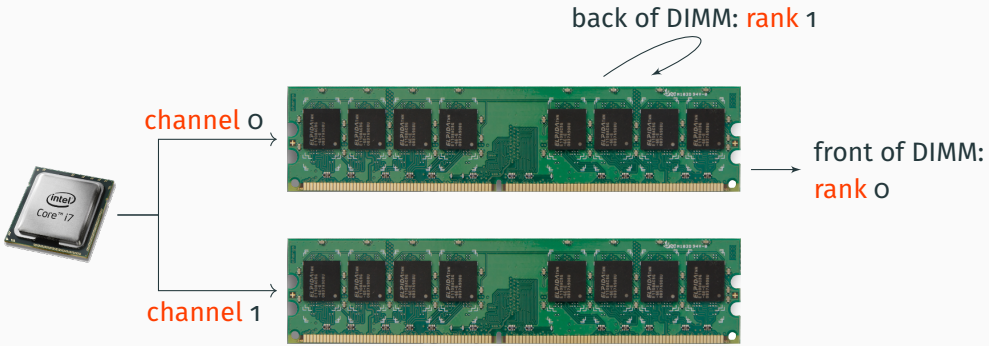
Software-based fault attacks

DRAM organization

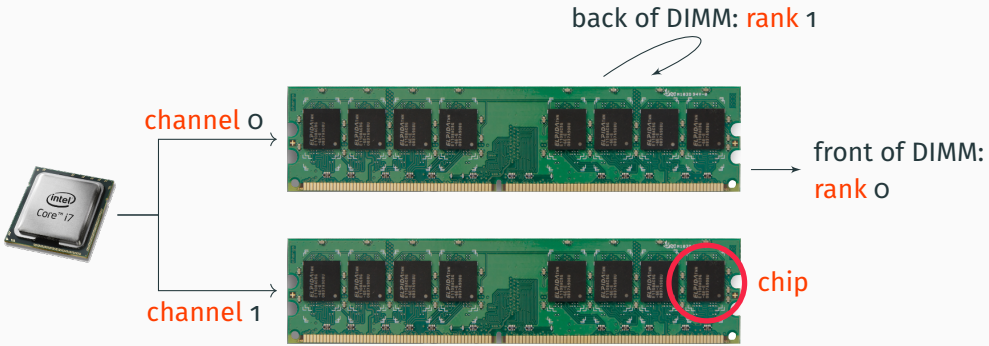
DRAM organization



DRAM organization

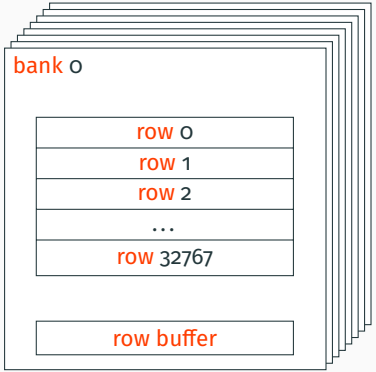


DRAM organization



DRAM organization

chip



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

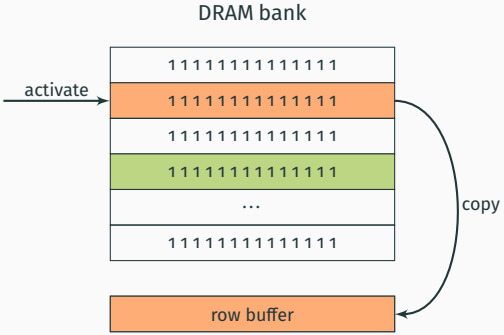
Rowhammer

"It's like breaking into an apartment by repeatedly slamming a neighbor's door until the vibrations open the door you were after" – Motherboard Vice

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

Rowhammer

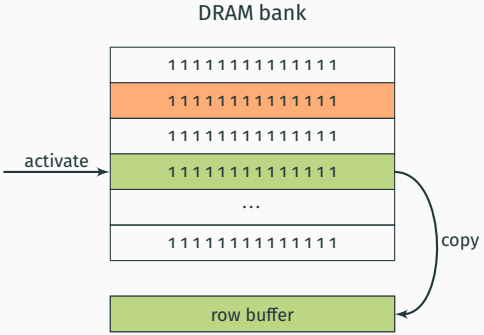
"It's like breaking into an apartment by repeatedly slamming a neighbor's door until the vibrations open the door you were after" – Motherboard Vice



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

Rowhammer

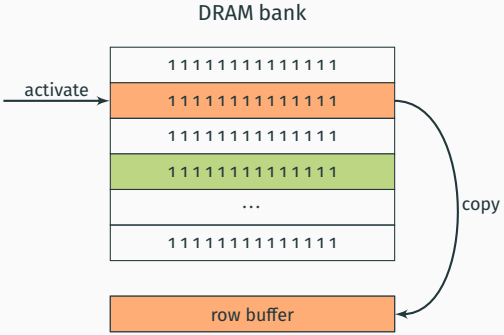
"It's like breaking into an apartment by repeatedly slamming a neighbor's door until the vibrations open the door you were after" – Motherboard Vice



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

Rowhammer

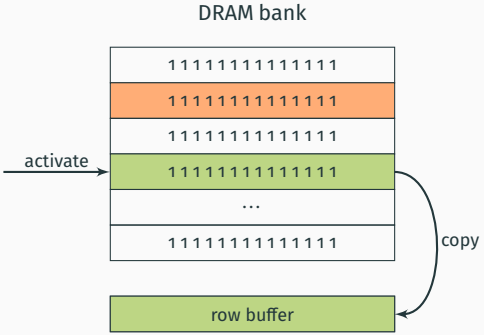
"It's like breaking into an apartment by repeatedly slamming a neighbor's door until the vibrations open the door you were after" – Motherboard Vice



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

Rowhammer

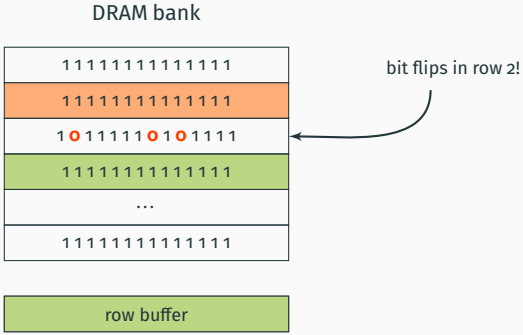
"It's like breaking into an apartment by repeatedly slamming a neighbor's door until the vibrations open the door you were after" – Motherboard Vice



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

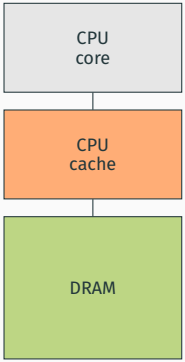
Rowhammer

"It's like breaking into an apartment by repeatedly slamming a neighbor's door until the vibrations open the door you were after" – Motherboard Vice



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

Impact of the CPU cache



- only non-cached accesses reach DRAM
- original attacks use `clflush` instruction

→ flush line from cache

→ next access will be served from DRAM

How to reach DRAM (x86 and ARM)?

1. `clflush` instruction $\rightarrow$ original paper (Kim et al.)
2. non-temporal accesses (Qiao et al.)
3. cache eviction $\rightarrow$ Prime+Probe (Gruss et al., Aweke et al., Frigo et al.)
4. uncached memory (van der Veen et al.)

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

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

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

P. Frigo et al. “Grand Pwning Unit: Accelerating Microarchitectural Attacks with the GPU”. In: *S&P*. 2018.

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

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

What works on ARM?

1. ARMv7 flush instruction is privileged ✗
2. ARMv8 non-temporal stores are still cached in practice ✗
3. cache eviction seems to be too slow, except when using GPUs
   → also works in browsers using WebGL ✓
4. apps can use `/dev/ion` for uncached, physically contiguous memory, without any privilege or permission needed (since Android 4.0) ✓

Conclusion

Conclusion

- some differences in techniques used for side-channel attacks and fault attacks on x86 and ARM…
- … but nothing fundamentally different
- attacks based on optimizations
- how to get rid of the attacks while keeping the optimizations?