Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer attack

Scalar Lab 김지현

A Practical ECCploit solution

- Prerequisite

1. Knowledge of ECC function
2. A side channel to observe bit flips
3. Ability to control/compose bit flips via data patterns in aggressor rows

-> end-to-end Rowhammer exploits on ECC-equipped systems

ECCploit three phases

- First, template memory to find correctable bits
- Second, try to combine multiple of these bit flips → Create error patterns that the ECC function cannot detect
- Finally, use these patterns to launch exploits

Which victims?

- Page table entries
- RSA public keys
- Binary code

TABLE I: Target systems.

ID	Manufacturer	CPU model	Microarchitecture
AMD-1	AMD	Opteron 6376	Bulldozer (15h)
Intel-1	Intel	Xeon E3-1270 v3	Haswell
Intel-2	Intel	Xeon E5-2650 v1	Sandy Bridge
Intel-3	Intel	Xeon E5-2620 v1	Sandy Bridge

A. Templating correctable errors

A. Templating correctable errors

- Templating phase

- Probe memory to see if we can safely trigger bit flips using Rowhammer
- Only cause error that ECC function can correct automatically

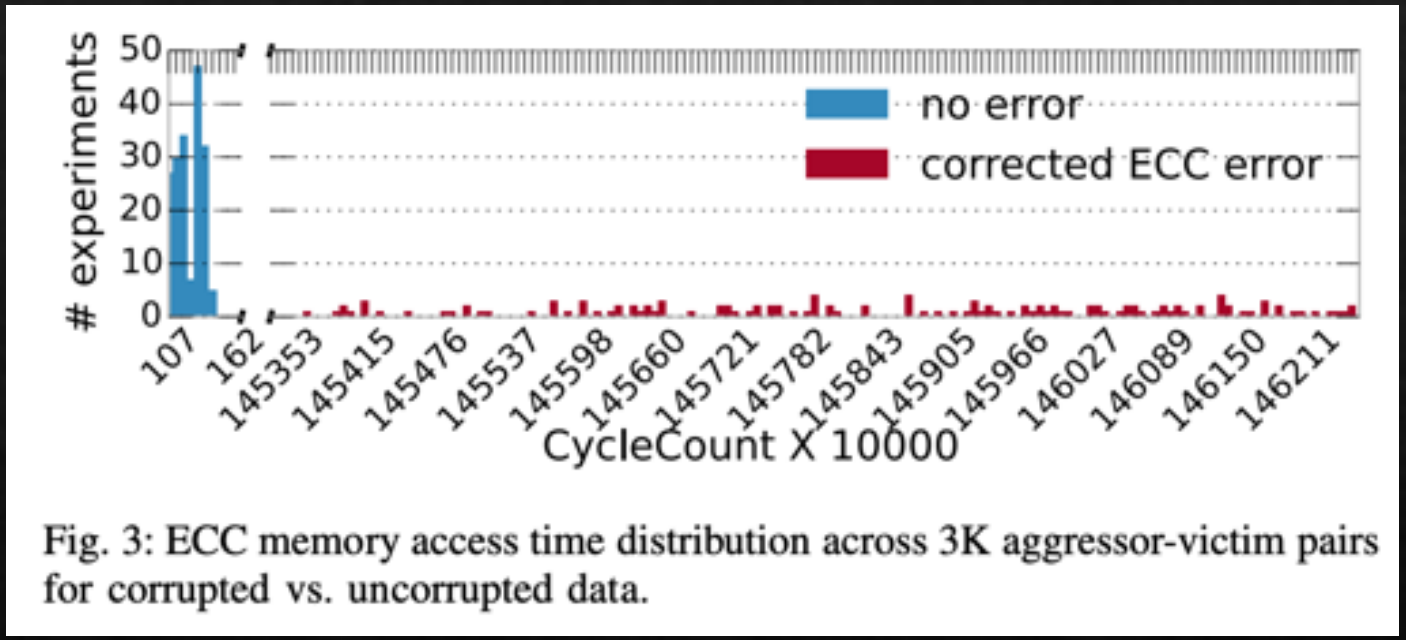


Fig. 3: ECC memory access time distribution across 3K aggressor-victim pairs for corrupted vs. uncorrupted data.

Observe bit flips through side channel

A-1. Target address selection

- A double-sided Rowhammer : 2 aggressor rows targeting 1 victim row between

- Start with a list of potential aggressor locations(a1, a2) and victim addresses(v)

(Same bank, different rows)

- Mapping between virtual and physical address(use existing reverse engineering)
- However, still can attack without this information

A-2. Pattern selection

- Detect usable tuples of aggressor-victim-aggressor(a1, v, a2)

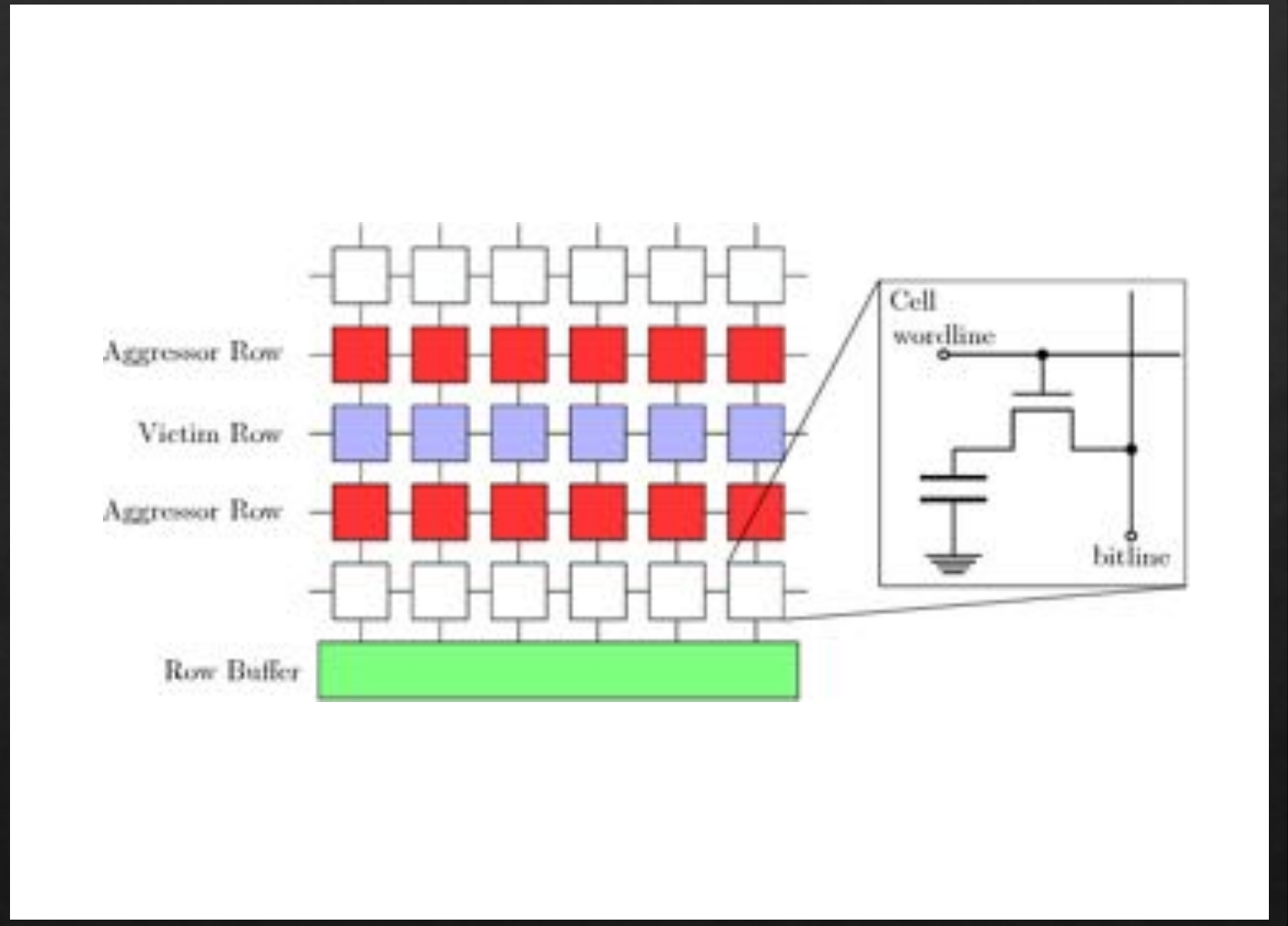
- Crash-free templating strategy (only make correctable ECC errors)
  - Hamming distance ≤ number of errors E that ECC algorithm can correct

- Check for directions in the bit flip(0->1 or 1->0) due to resulting striping patterns

Striping patterns?

※ 4 possible data patterns

1. **0/1-stripe**
2. **1/0- stripe**
3. 0-uniform
4. 1-uniform



A-3. Search strategy

- Target all the words in the victim row at the same time during each hammering attempt

“ if the ECC corrects single bit errors, we hammer first with bit patterns in the aggressor and victim rows such that aggressors and victim differ only in the most significant bit of each of the ECC word in the row, then with patterns that differ only in the next bit and so on..”

Size (bits)	Size (bits)	Size (bits)	Bit Overhead
8	4	12	50.0%
16	5	21	31.3%
32	6	38	18.8%
64	7	71	10.9%
128	8	136	6.3%

ECC word?

+ control bit/ data bit

A-3. Search strategy

- Read from the entire victim row at once (for each trial)
- Use side channel to detect bit flips anywhere in the row
- Exploits composability of bit flips (batch processing) -> efficient templating

A-3. Search strategy

- If detect bit flips, hammer tuple a few more times -> Identify flipping ECC words

- (pseudo) binary search :omit stripe patterns in words we are not testing
  : until reproduce the bit flips on one or more words

- repeat the entire process twice for each tuple (2 possible stripe patterns)

- identify vulnerable bits in both directions (1->0 or 0->1) for all tuples in memory

A-3. Search strategy

- Note down all the vulnerable **1-bit templates** with
  - corresponding (a1, v, a2) tuple
  - ECC word
  - word offset
  - direction of the bit flip in the victim row

B. Combining bit flips

B. Combining bit flips

- What we know: ECC algorithm, 1-bit templates(inducing correctable bit flips)

- Goal: Combine multiple bit flips in a single ECC word

  → **Produce new words that escape ECC detection**

B. Combining bit flips

- Step

1. group together all the 1-bit templates(with same aggressor rows, victim row, direction, ECC word) in a template group

2. Generate possible flipped words (induced via Rowhammer) that that bypass the target ECC algorithm

What's in template group

- Combination of k 1-bit templates that would induce k bit flips that result in

[P4] corruption ECC does not correct

[P1], [P2], [P3] ECC does not detect

TABLE V: Error patterns that can circumvent ECC.

ID	Pattern	Config.	# flips	Flips location
AMD-1	$[\mathcal{P}_1]$	Ideal	3-BF-16	3 symbols, 1 in control bits
AMD-1	$[\mathcal{P}_2]$	Ideal	4-BF-16	Min. 2 symbols
Intel-1	$[\mathcal{P}_3]$	Ideal	4-BF-8	Min. 2 symbols
Intel-1	$[\mathcal{P}_4]$	Default	2-BF-8	Min. 2 symbols

Exploitable patterns (AMD-1)

- [P1] attacker requires at least 3 bit flips in 16 bytes(i.e., ECC word)
  - 1 of the bit flips is in the control bit
  - other 2 bits flips in 2 distinct symbols (at least 8 bits apart)

- [P2] targeting data bits alone, 4 bit flips in at least 2 distinct symbols in an ECC word

Exploitable patterns (Intel-1)

- [P3] 4 bit flips in at least 2 distinct symbols (i.e., 4 bits apart)
- [P4] 2 bit flips in distinct symbols in an ECC word (uncorrectable errors -> but X crash in intel)

Results of templating step

Overall 265 templates available

(without ECC 1 minute)

8 weeks 6 days 4 hours

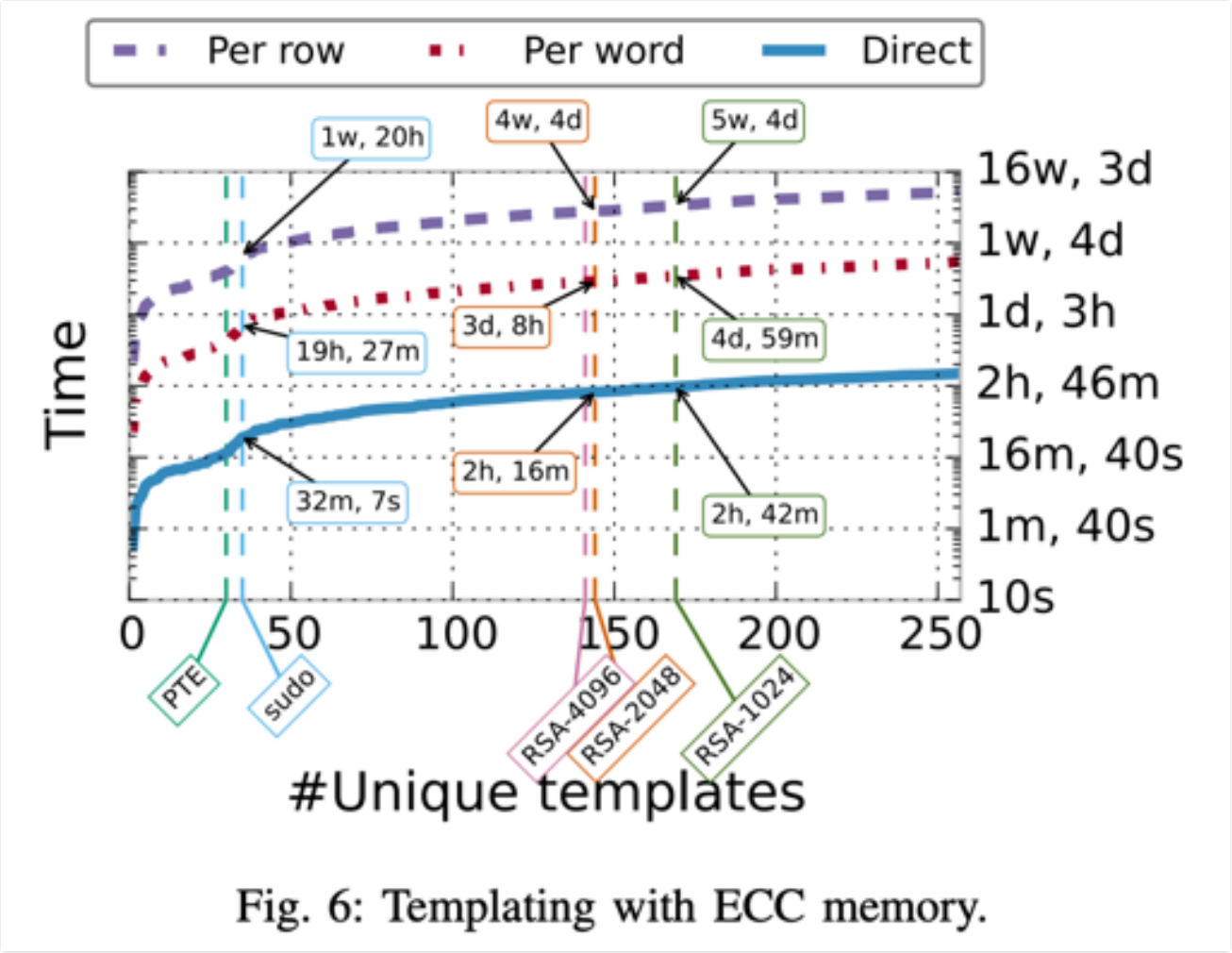


Fig. 6: Templating with ECC memory.

Conclusion of templating

- *"Attacker can run code on demand on the victim machine and complete a templating step of hours or even days in complete isolation without interfering with the rest of the system"*

- After templating, similar to existing non-ECC exploits

C. Exploitation

C. Exploitation

- What we have: ECC-aware templates

- Steps of practical exploit

1. Massaging the target data onto the vulnerable location
2. Setting the corresponding aggressor bit values as dictated by templates
3. Hammering to reliably reproduce the (composed) bit flips on the victim data

Key difference of ECCploit(Challenge)

- Difference/challenge with existing Rowhammer exploits

1. The number of useful templates is much lower
   (template: combination of bit flips to bypass ECC)
2. ECC templates corrupt multiple bits → complicate existing Rowhammer attacks

C-1. Page Table Entry(PTE) ECCploit (Result)

- 6.15% of 265 templates are exploitable (rest crash system)
- Success rate (even with imperfect page table spraying strategy)

- 39.9% map unauthorized memory pages

-2.5% with page table page

- Fails to modify any PTE

- victim PTE does not always have the target bits set in the direction of chosen template
- 5% success of similar non-ECC (∴ ECCploit strategy little impact on success)

C-2. RSA ECCploit (Result)

- 1337 randomly generated RSA keys (1024, 2048, 4096bit)
- 265 templates could only mutate given
  - 1024 bit key 2.8 times
  - 2048 bit key 5.5 times
  - 4096 bit key 9.4 times
- Factorization
  - 45.1% 1024 bit keys
  - 37% 2048 bit keys
  - 28.7% 4096 bit keys

C-3. Opcode modification ECCploit (Result)

- Template #36 flips bit 0 and 5 of a single byte changing

conditional branch instruction→ mov instruction

(inst jne $8fa0 at offset 0xbdc0) to (0x1da (%rbp), %eax))

Conclusion

It’s harder but possible!

For more details....

https://medium.com/@Anna_IT/rowhammer-공격-대응을-위한-ecc-메모리의-효과는-error-correcting-code의-실효성-검증