Rowhammer.js: A Remote Software-Induced Fault Attack in Javascript

Daniel Gruss, Clementine Maurice and Stefan Mangard

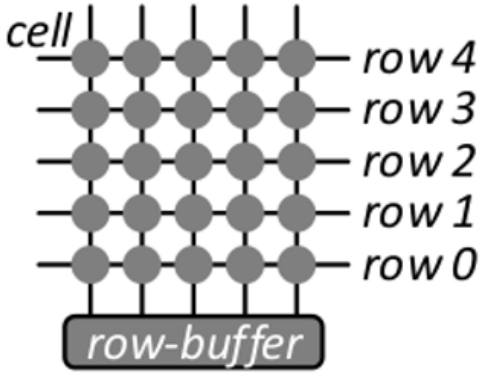
Graz University of Technology, Austria

Rowhammer bug (I)

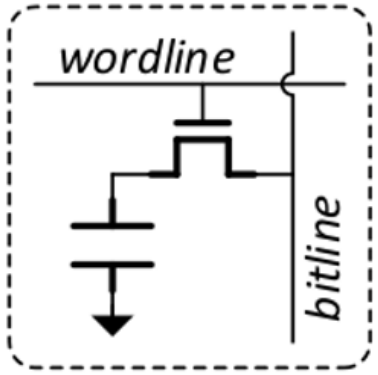
- Different DRAM cells can interfere with each other
  - *''When a DRAM row is opened and closed repeatedly, it can induce disturbance errors in adjacent rows"*

```
start:
    mov (X), %eax
    mov (Y), %ebx
    clflush (X)
    clflush (Y)
    mfence
jmp start
```

- X and Y addresses are mapped into the same bank but different rows
- The MC has to open and close the rows repeatedly



a. Rows of cells



b. A single cell

[1]: "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors" which appeared in ISCA in June 2014

Rowhammer bug (II)

The key findings of the Rowhammer bug are:

- **Errors are widespread:** Tested 129 modules from 3 different manufacturers
- **Errors are symptoms of charge loss:** A given DRAM module can experience a bit flip in the direction ‘1->0’ or ‘0->1’ depending on the orientation of DRAM
- **Errors occur in adjacent rows**
- **Errors are access pattern dependent**

[2]: Exploiting the DRAM rowhammer bug to gain kernel privilege, Seaborn, Dullien

Contributions of the paper

- Prove that they can perform Rowhammer without clflush
- They provide a cache eviction space exploration technique
- They implement Rowhammer attack using native code, on four different processors
- They demonstrate a Javascript Rowhammer implementation through a web browser

Steps to Rowhammer attack

1. Find 2 addresses in different rows of the same bank
2. Evict and reload these addresses fast
3. Search for an exploitable bit flip
4. Exploit the bit flip

They focus on the first two steps

- They find parts of the physical addresses taking advantage of large arrays
- They introduce a method for identifying cache eviction strategies

Cache eviction strategies

- Replacement policies are not always trivial
- They need to identify *congruent* addresses and the eviction strategy of the cache to find an access pattern
- A good access pattern with high eviction rate -> replacement for *clflush*
- Distinguish between static/dynamic eviction set and access pattern
  - Eviction set: knowledge of address mapping and physical addresses
  - Access pattern: memory address accesses

Finding an eviction strategy

- Offline and Online phase
  - **Offline phase:** Explore the parameter space to find the best eviction strategy for a set of systems
  - **Online phase:** Try to attack a system where you have no privileges. Use the offline phase information to gain knowledge for the system. Have a plan B in case this fails
- To find an eviction set and an access pattern they care about addresses in the same cache set

Pattern Testing

```
1 for (s = 0; s <= S-D; s += L)
2   for (c = 0; c <= C; c += 1)
3     for (d = 0; d <= D; d += 1)
4       *a[s+d];
```

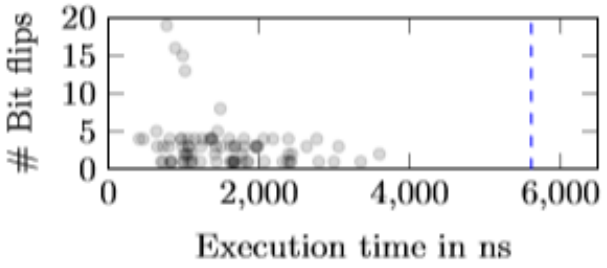
Listing 1: Eviction loop for pattern testing.

- **C:** number of accesses to each memory address
- **D:** number of different memory addresses accessed
- **L:** the loop step size
- **S:** the eviction set size

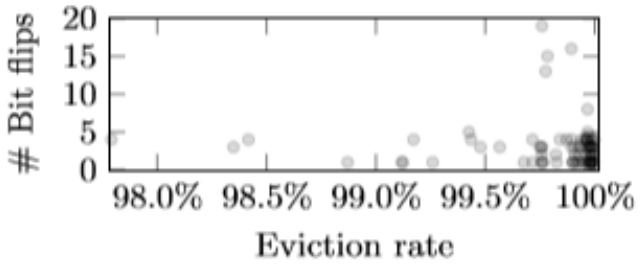
- They tested 6 dimensions of C, D, L and 23 set sizes -> more than 6 days

Offline Phase

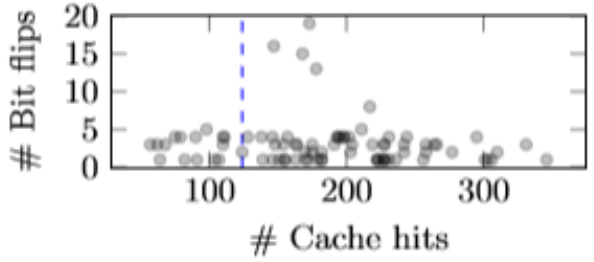
- No time constraints. Tested a specific set of addresses
- Try to find the best eviction strategy for a given machine



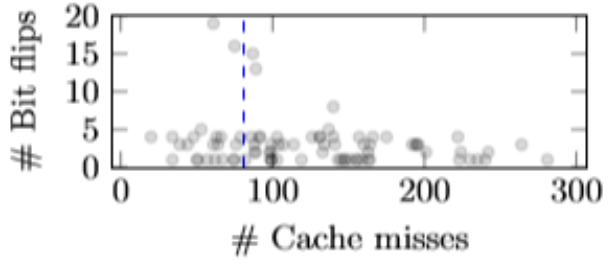
(a) Low execution time is better.



(b) High eviction rate is better. Average over all eviction strategies is 73.96%.



(c) Number of cache hits is not a good criteria for bit flips.



(d) Number of cache misses is not a good criteria for bit flips.

C	D	L	S	Accesses	Hits	Misses	Time (ns)	Eviction
-	-	-	-	-	2	2	60	99.9999%
5	2	2	18	90	34	4	179	99.9624%
2	2	1	17	68	35	5	180	99.9820%
2	1	1	17	34	47	5	191	99.8595%
6	2	2	18	108	34	5	216	99.9365%
1	1	1	17	17	96	13	307	74.4593%
4	2	2	20	80	41	23	329	99.7800%
1	1	1	20	20	187	78	934	99.8200%

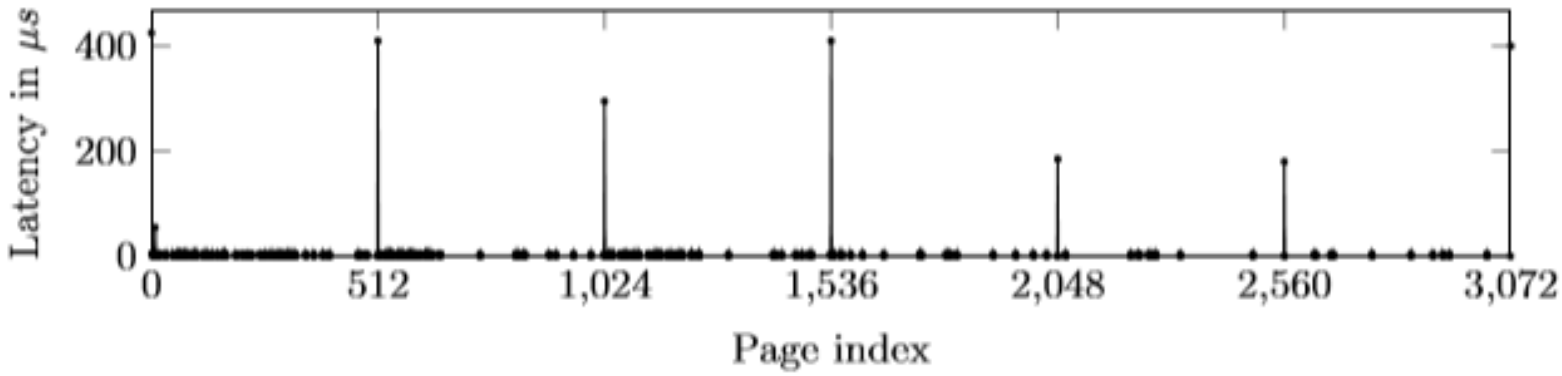
Online Phase

- **Assumption-based attack**
  - First test the results of the offline phase -> timing tests
  - Define a desired eviction rate threshold
  - Use the mapping algorithms as found in previous works for eviction set
  - Large arrays are allocated on large pages -> compute the addressing function as documented in previous papers, find addresses in the same cache slice/set
- **Fall-back attack**
  - If the assumption based attack fails
  - Compute a dynamic eviction set with a static access pattern
    - Add addresses to the eviction set multiple times
    - When an eviction threshold is reached replace addresses that do not lower the eviction rate with addresses in the set -> lower execution time
    - Remove randomly addresses that do not lower the eviction rate

Eviction based Rowhammer

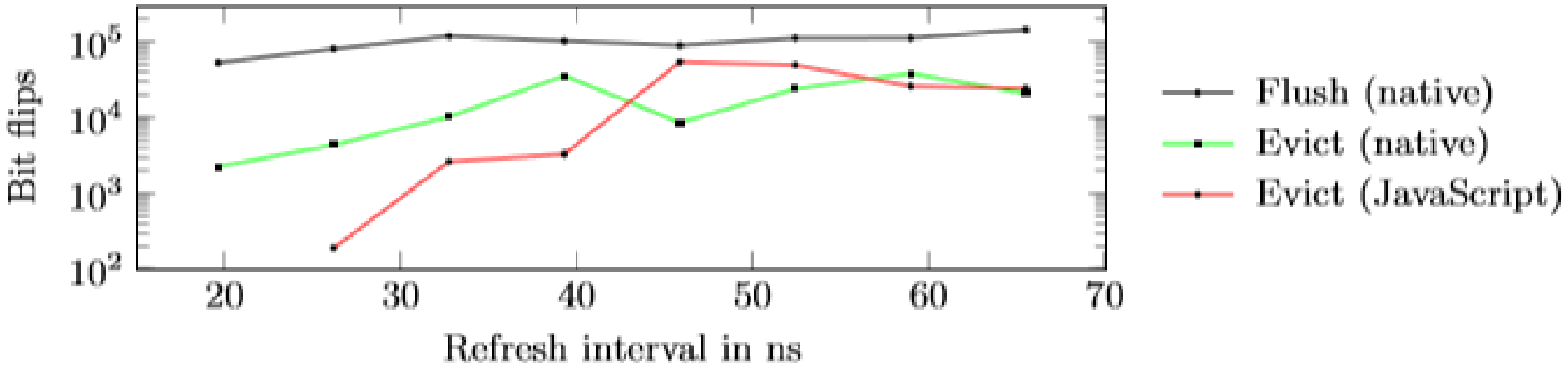
- Native code
  - Use the best eviction strategy they found to replace *clflush*
  - Eviction sets are precomputed manually using the address mapping from previous work

- Javascript
  - They use 2MB pages and arrays are 1MB aligned



- The use of 2MB pages gives them knowledge of the 21 LSBs
- Attacked Firefox using exact physical addresses as in native code
- Several congruent addresses per 2MB pages

Bit flips vs Refresh Rate



- *Clflush* achieves the highest rate of bit flips
- Decreasing the refresh interval decreases bit flips

Limitations & countermeasures

Limitations

- They are limited to the fact that 2MB pages are used
- Single-sided Rowhammer has a lower probability for a bit flip
- Double-sided Rowhammer has a lower probability to flip an exploitable bit
- They know the mapping of physical addresses to DRAM cells
- Physical addresses to L3 cache slices

Countermeasures

- Increase refresh rate
- Allocate kernel pages in a different memory frame
- ECC
- Refresh rows according to neighboring row access count