SledgeHammer: Amplifying Rowhammer via Bank-level Parallelism

Ingab Kang, *University of Michigan;* Walter Wang and Jason Kim, *Georgia Tech;* Stephan van Schaik and Youssef Tobah, *University of Michigan;* Daniel Genkin, *Georgia Tech;* Andrew Kwong, *UNC Chapel Hill;* Yuval Yarom, *Ruhr University Bochum*

https://www.usenix.org/conference/usenixsecurity24/presentation/kang

USENIX Security '24 Artifact Appendix: SledgeHammer: Amplifying Rowhammer via Bank-level Parallelism

Ingab Kang
University of Michigan
igkang@umich.edu

Walter Wang
Georgia Tech
walwan@gatech.edu

Jason Kim
Georgia Tech
nosajmik@gatech.edu

Stephan van Schaik
University of Michigan
stephvs@umich.edu

Youssef Tobah
University of Michigan
ytobah@umich.edu

Daniel Genkin
Georgia Tech
genkin@gatech.edu

Andrew Kwong
UNC Chapel Hill
andrew@cs.unc.edu

Yuval Yarom
Ruhr University Bochum
yuval.yarom@rub.de

A Artifact Appendix

A.1 Abstract

This artifact is an implementation of multi-bank hammering in our paper. It is based on the TRRespass repository and designed to test bitflips on Intel systems with DDR4 memory. The code is expected to show a near-linear increase in the number of total bitflips with the increase of hammered banks and a jump in the effectiveness of the hammering pattern when the number of hammered banks is increased from one to two.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Running this program should not pose any security, privacy, or ethical concerns that we know of. By design, bitflips found by the code are contained in address spaces within the program. However, it may be possible that the program may flip a random bit in the system and make the system to be unstable.

A.2.2 How to access

The code is avaliable at: Github Page Link

A.2.3 Hardware dependencies

Running this program requires a 6th to 10th-generation Intel processor with a DDR4 compatible mainboard. The installed DDR4 memory must be susceptible to TRRespass.

A.2.4 Software dependencies

The code requires gcc and Make to compile. DRAM addressing function in *src/main.c* must be configured according to the DRAM configuration. We include the addressing function for 8GB x8 single DIMM, single channel and 16GB x8 two DIMMs, dual channel configuration for 6th to 10th-generation Intel processors. The addressing function can be configured through *src/main.c*. Additional addressing functions for different configurations can be discovered using DRAMA,

The OS should be linux-based and the code requires 1GB hugepage to be available or transparent hugepages be set to *madvise* or *always* in the OS.

A.2.5 Benchmarks

None.

A.3 Set-up

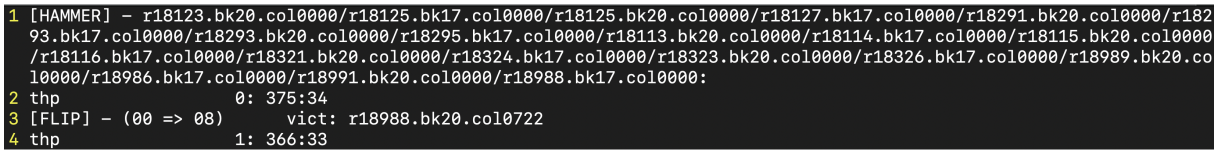
The machine running the program should satisfy all hardware and software dependencies from the previous section. While running the program, it is advised that no other program is running alongside it, as it may interfere with the hammering process.

A.3.1 Installation

With the test machine setup, *cd* into *multibank_hammer* and run *make*. Then, run *sudo ./hugepage.sh* to mount the 1 GB huge page.

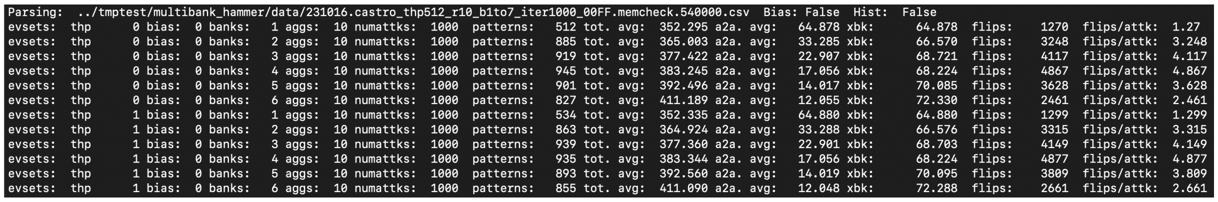
A.3.2 Basic Test

After compiling the code, run *sudo ./obj/tester -v*. In our configuration, we were able to see a bitflip within the first minute. If a bitflip is found, there will be a command line output notifying the user of the flip direction and the row, bank, and column of the flipped bit. The program will terminate after 1000 different hammering iterations.



```
1 [HAMMER] - r18123.bk20.col0000/r18125.bk17.col0000/r18125.bk20.col0000/r18127.bk17.col0000/r18291.bk20.col0000/r182
  93.bk17.col0000/r18293.bk20.col0000/r18295.bk17.col0000/r18113.bk20.col0000/r18114.bk17.col0000/r18115.bk20.col0000
  /r18116.bk17.col0000/r18321.bk20.col0000/r18324.bk17.col0000/r18323.bk20.col0000/r18326.bk17.col0000/r18989.bk20.co
  l0000/r18986.bk17.col0000/r18991.bk20.col0000/r18988.bk17.col0000:
2 thp                0: 375:34
3 [FLIP] - (00 => 08)      vict: r18988.bk20.col0722
4 thp                1: 366:33
```

Figure 1: Example of output.



```
Parsing:  ../tmptest/multibank_hammer/data/231016.castro_thp512_r10_b1to7_iter1000_00FF.memcheck.540000.csv  Bias: False  Hist:  False
evsets:  thp      0 bias:  0 banks:   1 aggs:  10 numattks:  1000  patterns:   512 tot. avg:  352.295 a2a. avg:   64.878 xbk:      64.878  flips:    1270  flips/attk:  1.27
evsets:  thp      0 bias:  0 banks:   2 aggs:  10 numattks:  1000  patterns:   885 tot. avg:  365.003 a2a. avg:   33.285 xbk:      66.570  flips:    3248  flips/attk:  3.248
evsets:  thp      0 bias:  0 banks:   3 aggs:  10 numattks:  1000  patterns:   919 tot. avg:  377.422 a2a. avg:   22.907 xbk:      68.721  flips:    4117  flips/attk:  4.117
evsets:  thp      0 bias:  0 banks:   4 aggs:  10 numattks:  1000  patterns:   945 tot. avg:  383.245 a2a. avg:   17.056 xbk:      68.224  flips:    4867  flips/attk:  4.867
evsets:  thp      0 bias:  0 banks:   5 aggs:  10 numattks:  1000  patterns:   901 tot. avg:  392.496 a2a. avg:   14.017 xbk:      70.085  flips:    3628  flips/attk:  3.628
evsets:  thp      0 bias:  0 banks:   6 aggs:  10 numattks:  1000  patterns:   827 tot. avg:  411.189 a2a. avg:   12.055 xbk:      72.330  flips:    2461  flips/attk:  2.461
evsets:  thp      1 bias:  0 banks:   1 aggs:  10 numattks:  1000  patterns:   534 tot. avg:  352.335 a2a. avg:   64.880 xbk:      64.880  flips:    1299  flips/attk:  1.299
evsets:  thp      1 bias:  0 banks:   2 aggs:  10 numattks:  1000  patterns:   863 tot. avg:  364.924 a2a. avg:   33.288 xbk:      66.576  flips:    3315  flips/attk:  3.315
evsets:  thp      1 bias:  0 banks:   3 aggs:  10 numattks:  1000  patterns:   939 tot. avg:  377.360 a2a. avg:   22.901 xbk:      68.703  flips:    4149  flips/attk:  4.149
evsets:  thp      1 bias:  0 banks:   4 aggs:  10 numattks:  1000  patterns:   935 tot. avg:  383.344 a2a. avg:   17.056 xbk:      68.224  flips:    4877  flips/attk:  4.877
evsets:  thp      1 bias:  0 banks:   5 aggs:  10 numattks:  1000  patterns:   893 tot. avg:  392.560 a2a. avg:   14.019 xbk:      70.095  flips:    3809  flips/attk:  3.809
evsets:  thp      1 bias:  0 banks:   6 aggs:  10 numattks:  1000  patterns:   855 tot. avg:  411.090 a2a. avg:   12.048 xbk:      72.288  flips:    2661  flips/attk:  2.661
```

Figure 2: Example of the parser output.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): The provided multi-bank hammering pattern exhibits a jump in the number of bitflips per bank per hammering attempt when increasing the number of hammered banks from one to two.

(C2): The total number of bitflips per hammering attempt will increase almost linearly with the increase in the number of hammered banks when increasing from two. The optimal number of banks would differ per configured system. In our case, the number of bitflips peaked at 4 banks.

A.4.2 Experiments

By default, the code is configured to hammer 10 aggressors per bank and increase the number of hammered banks from 1 to 6, hammering 1000 iterations per bank configuration. On our machine, this took around 2 hours to run.

Preparation: Set the DRAM addressing function in *src/main.c* according to the system configuration and *num_aggs* in *hammer-suite.c* to the optimal number of aggressors of the installed DDR4 DIMM and run *make*.

Execution: run *sudo ./obj/tester -v*. Parse the result by running *python3 ./parse_result.py data/test.csv -v*.

Results: Figure 1 is the an example of the program output. Line 1 shows the aggressor rows in the order that they are hammered. line 2 and 4 is the hammering pattern (thp), the 0 and 1 after the pattern indicate whether we're looking for 0 to 1 flips or 1 to 0 flips, respectively. The number after the colon is the total time it took for the hammering pattern (ms) and the time between two ACTs to rows across banks (ns). Note that because this is across banks, the time between ACTs would be smaller as ACTs are parallelized. Finally, line 3 is the output when a bitflip is found. It indicates the flip direction and the location of the flipped bit. This information, aggressor rows, timing information, and the bitflips, are also saved as a file in *data/test.csv* by default.

Figure 2 is an exemplary output of the parser. *thp 0* are the results for 0 to 1 flips and *thp 1* are the results for 1 to 0 flips. *numattks* is the total number of different hammering attempts, *patterns* is the number of patterns that was able to flip bits, *tot. avg* is the average time in ms to run the hammering pattern, *a2a* is the average time between activates (across banks), *xbk* is the average time between activates within a bank. *flips* is the total number of flips across all iterations, and *flips/attk* is the total number of flips divided by the number of iterations.

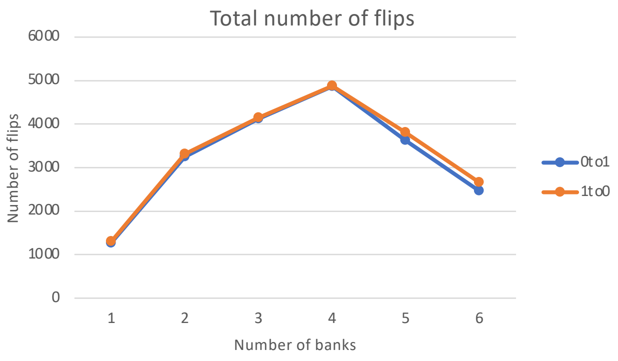


Figure 3: Total number of flips plot.

If you plot the total number of flips, it should look like Figure 3, where the number of flips increases with the number of banks. In Figure 3, the peak was with 4 banks, a 2.85x increase in total number of bitflips compared to 1 bank. The peak number of banks and the total number of bitflips may differ per configuration.

Figure 4 is the total number of flips divided by the hammering iterations divided by hammered banks. This is to measure how effective the hammering pattern is at flipping bits. The plot should show a jump when increasing the number of banks from one to two and steadily decline afterwards. In Figure 4

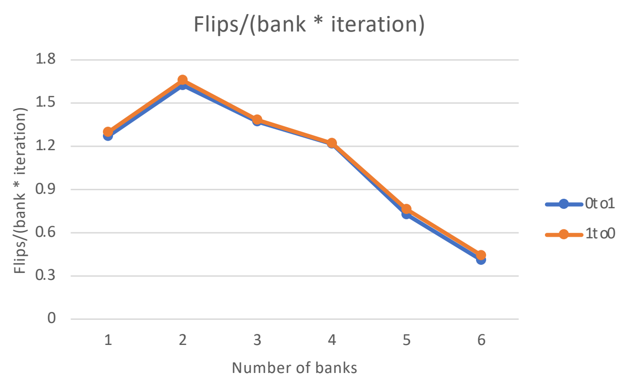


Figure 4: Total number of flips / (number of iterations * number of banks).

the effectiveness jumped by 1.2x compared to a single bank. The increase in effectiveness may differ per configuration.

A.5 Notes on Reusability

While multi-bank hammering was only tested with n-sided hammering, the same principles that increased the number of flips should apply to other hammering patterns, such as Blacksmith. This would allow the user to create more powerful hammering patterns.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at `https://secartifacts.github.io/usenixsec2024/`.