Rowhammering Storage Devices

Tao Zhang, Boris Pismenny, Donald E. Porter, Dan Tsafrir, Aviad Zuck
Explore the feasibility of rowhammering the DRAM inside an SSD, using *only standard storage commands*.

Understand whether the *small embedded system within the device* is vulnerable to the same rowhammer attack like a “big” system.
DRAM Overview

- DRAM cell array
  - Addressed by row/column
DRAM Overview

- DRAM cell array
  - Addressed by row/column
  - Bits stored as electric charges
### DRAM Overview

- **DRAM cell array**
  - Addressed by row/column
  - Bits stored as electric charges
  - Cells lose charge over time

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Bank**

- **columns**
## DRAM Overview

<table>
<thead>
<tr>
<th>Bank</th>
<th>columns</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>row n</td>
<td><img src="cell_array.png" alt="Cell Array" /></td>
</tr>
<tr>
<td>row n+1</td>
<td><img src="row_n_plus_1.png" alt="Row n+1" /></td>
</tr>
<tr>
<td>row n+2</td>
<td><img src="row_n_plus_2.png" alt="Row n+2" /></td>
</tr>
<tr>
<td></td>
<td><img src="row_n_plus_3.png" alt="Row n+3" /></td>
</tr>
</tbody>
</table>

- **DRAM cell array**
  - Addressed by row/column
  - Bits stored as electric charges
  - Cells **lose charge** over time
    - Refresh rows in intervals (64ms)
Rowhammer attack

- **Disturbance error**
  - **Aggressor row**: Repeatedly open/close
  - **Victim(adjacent) row**: charge leaks faster
  - Leak faster than refresh, *bitflip* happens

<table>
<thead>
<tr>
<th></th>
<th>column 1</th>
<th>column 2</th>
<th>column 3</th>
<th>column 4</th>
<th>column 5</th>
<th>column 6</th>
<th>column 7</th>
<th>column 8</th>
<th>column 9</th>
<th>column 10</th>
<th>column 11</th>
<th>column 12</th>
<th>column 13</th>
<th>column 14</th>
<th>column 15</th>
<th>column 16</th>
<th>column 17</th>
<th>column 18</th>
<th>column 19</th>
<th>column 20</th>
</tr>
</thead>
<tbody>
<tr>
<td>row n</td>
<td>0 0 1 0 1 0 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>row n+1</td>
<td>0 0 1 0 1 0 1 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>row n+2</td>
<td>1 1 1 1 1 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Rowhammer attack

- Disturbance error
  - Aggressor row: Repeatedly open/close
  - Victim (adjacent) row: charge leak faster
  - Leak faster than refresh, bitflip happens

- Double-sided rowhammer

Bank

```
<table>
<thead>
<tr>
<th>columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>row n</td>
</tr>
<tr>
<td>0 0 1 0 1 0 1 1</td>
</tr>
<tr>
<td>row n+1</td>
</tr>
<tr>
<td>0 0 0 1 1 0 1 0</td>
</tr>
<tr>
<td>row n+2</td>
</tr>
<tr>
<td>1 1 1 1 1 1 1 1</td>
</tr>
</tbody>
</table>
```

H: Head (row n)

[loop:

```c
mov (addr1), %eax
mov (addr2), %ebx
clfush (addr1)
clfush (addr2)
mfence
jmp loop
```]
Outcomes of Rowhammer Attack

Data corruption

Information leak

Privilege escalation
Trends in DRAM Technology

- Technology node size (nm): Older DRAM > Newer DRAM
- Density: Older DRAM < Newer DRAM
- # accessing rate before bit flips: Older DRAM > 150K access/s > Newer DRAM

DRAM is becoming increasingly more vulnerable to rowhammer attack!
How to Rowhammer SSD?

SSD works as a black box, little internals known

- Reveal internals by reverse engineering
- Not able to run rowhammer code directly
- Find an indirect way for rowhammering
SSD Reverse Engineering

- SSD as a “computer system”
  - 3 core Cortex-R4 ARMv7 CPU
  - 512MiB LPDDR3 DRAM
  - 120GiB NAND flash

- SSD DRAM
  - FTL runtime code & data
  - Buffer for incoming I/O commands
  - Logical-to-physical mapping table (L2P)
    - Linear table, hash table
  - DRAM accesses are uncached

Can we rowhammer it?
L2P Table

<table>
<thead>
<tr>
<th>L2P Table</th>
<th>LBA 512</th>
<th>...</th>
<th>LBA 767</th>
</tr>
</thead>
<tbody>
<tr>
<td>LBA 256</td>
<td>...</td>
<td>LBA 511</td>
<td></td>
</tr>
<tr>
<td>LBA 0</td>
<td>...</td>
<td>LBA 767</td>
<td></td>
</tr>
</tbody>
</table>

Rowhammer L2P by using SSD as intended (read)
Rowhammering the L2P Table

Latest PCIe 4.0 NVMe SSDs provide ~1.5M IOPS, compare to decreasing access rate for rowhammer (150K access/s)
Outcomes of Rowhammering SSD?

- Data corruption
- Information leak
- Privilege escalation

Need SSD-oriented exploits to turn bitflips into meaningful results.
Ext4 Direct/Indirect Block Addressing

inode

Data block \( a \)

Indirect blk \( b \)

Data block

Crafted data block to mimic indirect blk

Not protected by checksum!

LBA → ?

12 entries
Attacker owned file

inode

Attacking the Ext4 Indirect Block

Indirect blk a

Data block b

Maliciously formed indirect blk

Data block c

LBA

original L2P mapping

redirected L2P mapping

L2P table

Data block that attacker has no access permission

13
Conclusions

- Threats of rowhammer attack extended to a new dimension, storage devices
  - What about other attacks targeting host-side hardware?
  - What are possible mitigations?
  - Do we have a more principled solution?
Discussions
*DRAM Overview

Channel

DIMM

Rank

Bank

DRAM Cell Array
**Attack Scenario: Cloud Server**

Unprivileged attacker process in victim system:
- Has **normal** access to owned files
  - read/write/create/delete

Attacker VM shares the same SSD with victim:
- Hardware pass-through to SSD partition
  - SRIOV or namespace

FTL manages L2P table and physical blocks, shared between VMs.
Accesses to **empty LBAs** (LBAs unwritten/TRIMed) are served **faster**
Attack Scenario: Single System

Attacker has a normal process in victim system

- Has unprivileged access to owned files
- Fast direct access to the underlying storage
  - O_DIRECT and libaio/io_uring

Attacker needs SSD that can serve read to non-empty LBAs fast enough
Mitigations

● Mitigations for host-side rowhammer attacks (e.g. ECC, TRR, cache)
  ○ Impact on performance, cost-efficiency, power-efficiency
  ○ Attacks circumventing these mitigations available

● Hardening the FTL
  ○ Stronger isolation between partitions/namespaces
  ○ Randomize FTL-internal structures

● Enforce block-level data integrity protection and encryption
  ○ Enforce extent tree addressing for Ext4
  ○ Can’t stop data corruption