Exploiting the DRAM row hammer bug to gain kernel privileges

Writer: MARK SEABORN @GOOGLE

Presenter: Jiwon Choi
Introduction

Exploit!

... without exploiting software bug
Row hammer

DRAM’s row

repeated accesses

DRAM chipset
DRAM Structure

DRAM chipset
DRAM Structure

(Diagram from ARMOR project, University of Manchester)
DRAM Structure

ex) 4GB memory = 2 ranks * 8 banks * 8K per row * 32768 rows
DRAM ?
Dynamic RAM !
DRAM is really dynamic!
DRAM row buffer

Row buffer
DRAM row buffer

Row buffer
Open
- raise **wordline** to high voltage

Row buffer
Open
- raise wordline to high voltage
- Connecting capacitor to bitline

Row buffer
Open
- raise wordline to high voltage
- Connecting capacitor to bitline

Row buffer
- Access to row buffer are fast
Open
- raise wordline to high voltage
- Connecting capacitor to bitline

Row buffer
- Access to row buffer are fast
Open
- raise wordline to high voltage
- Connecting capacitor to bitline
- DRO (Destructive Read Out)

Row buffer
- Access to row buffer are fast
<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<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>
<td></td>
</tr>
<tr>
<td></td>
<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>
<td></td>
</tr>
<tr>
<td></td>
<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>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Recharge
- Copy the row back

Row buffer
Cells are capacitor!
- They leak charge
- Cells should be periodically refreshed
- Refresh circuitry perform refresh cycle within the refresh time interval: 64ms
Introduction to rowhammer problems
Introduction to rowhammer problems
This “aggressor” row is repeatedly activated (hammered)
This “aggressor” row is repeatedly activated (hammered)

OPEN (voltage raise)
This “aggressor” row is repeatedly activated (hammered)
This “aggressor” row is repeatedly activated (hammered)

OPEN (voltage raise)

Result: These “victim” rows get bit flips
Bad Cells

- Randomly distributed
- Constantly flip when hammered
- Varies by DRAM module
  - % of rows with bad cells: Varies from 30% to 99.9%
03

Understand bit flipping by looking hammering code!
Challenge 1. Right way to flip bit. ①? ②?

Challenge 2. How to find pair of rows?
Challenge 1. Right way to flip bit. ①? ②?
Challenge 2. How to find pair of rows?
Challenge 1. Right way to flip bit. ①? ②?
Challenge 2. How to find pair of rows?
Challenge 1. Right way to flip bit. ①? ②?

Challenge 2. How to find pair of rows?
Challenge 1. Right way to flip bit. ①? ②?

Challenge 2. How to find pair of rows?
Challenge 1: Right way to flip bit. ①? ②?

Challenge 2. How to find pair of rows?
Challenge 1: Right way to flip bit. ①? ②?

Challenge 2: How to find pair of rows?

Random pick = 1/8
Bit flip code:

1. OPEN – CLOSE rows repeatedly
   pick 2 addresses: Same Bank Different Rows (SBDR)
2. CPU cache by clflush

```assembly
    code1a:
    mov (X), %eax
    mov (Y), %ebx
    clflush (X)
    clflush (Y)
    jmp codela
```
Bit flip code:

1. **OPEN – CLOSE** rows repeatedly
   - pick 2 addresses: Same Bank Different Rows (SBDR)
2. **CPU cache** by clflush

```assembly
codela:
mov (X), %eax
mov (Y), %ebx
clflush (X)
clflush (Y)
jmp codela
```
How to Exploit a bit flip

1. Native Client Sandbox
2. Linux Kernel
04

How to Exploit a bit flip

1. Native Client Sandbox
2. Linux Kernel
Native Client Sandbox

✓ Sandbox for running C/C++ “native code” on the web
✓ Used in Chrome
✓ Goal: make C/C++ code as safe as JavaScript
✓ In-process sandbox
  - Can’t call host OS’s syscalls
Native Client Sandbox

✓ Sandbox for running C/C++ “native code” on the web
✓ Used in chrome
✓ Goal: make C/C++ code as safe as JavaScript
✓ In-process sandbox
  - Can’t call host OS’s syscalls
  - Sandbox escape!
Challenges

1. Mark shellcode as executable
2. Jump to shellcode
Challenges

1. Mark shellcode as executable
2. Jump to shellcode

Allowed by NaCl’s validator

This conceals:

```
20ea0: 48 b8 0f 05 eb 0c f4 f4 f4
     movabs $0xf4f4f4f40ceb050f, %rax
```

```
20ea2: 0f 05     syscall
20ea4: eb 0c     jmp ... // Jump to next hidden instruction
20ea6: f4        hlt     // Padding
```
Challenges

1. Mark shellcode as executable
2. Jump to shellcode

Only allows “jmp *%rax” as part of this safe indirect jump sequence:

```
4c 01 f8  addq %r15, %rax  // Add %r15, the sandbox base address.
ff e0   jmp *%rax          // Indirect jump.
```
04

How to Exploit a bit flip

1. Native Client Sandbox
2. Linux Kernel
normal Linux process

1. Spray most of physical memory with page tables
2. Bit flip!

Kernel privilege escalation
Linux kernel exploit

...
Create shared memory
1. `mmap()` data file repeatedly

2. Spray memory page table

Map it multiple times
1. `mmap()` data file repeatedly

2. Spray memory page table
Row hammering
Got write access to page table!

Bit flipped in PTE

Virtual Address Space

Physical Memory

RW = 1
Got write access to page table!

Overwrite entry point of SUID-root executable (e.g. /bin/ping) to shell code

Privilege escalation!
Experimental results
<table>
<thead>
<tr>
<th>Laptop model</th>
<th>Laptop year</th>
<th>CPU family (microarchitecture)</th>
<th>DRAM manufacturer</th>
<th>Saw bit flip</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model #1</td>
<td>2010</td>
<td>Family V</td>
<td>DRAM vendor E</td>
<td>yes</td>
</tr>
<tr>
<td>Model #2</td>
<td>2011</td>
<td>Family W</td>
<td>DRAM vendor A</td>
<td>yes</td>
</tr>
<tr>
<td>Model #2</td>
<td>2011</td>
<td>Family W</td>
<td>DRAM vendor A</td>
<td>yes</td>
</tr>
<tr>
<td>Model #2</td>
<td>2011</td>
<td>Family W</td>
<td>DRAM vendor E</td>
<td>no</td>
</tr>
<tr>
<td>Model #3</td>
<td>2011</td>
<td>Family W</td>
<td>DRAM vendor A</td>
<td>yes</td>
</tr>
<tr>
<td>Model #4</td>
<td>2012</td>
<td>Family W</td>
<td>DRAM vendor A</td>
<td>yes</td>
</tr>
<tr>
<td>Model #5</td>
<td>2012</td>
<td>Family X</td>
<td>DRAM vendor C</td>
<td>no</td>
</tr>
<tr>
<td>Model #5</td>
<td>2012</td>
<td>Family X</td>
<td>DRAM vendor C</td>
<td>no</td>
</tr>
<tr>
<td>Model #5</td>
<td>2013</td>
<td>Family X</td>
<td>DRAM vendor B</td>
<td>yes</td>
</tr>
<tr>
<td>Model #5</td>
<td>2013</td>
<td>Family X</td>
<td>DRAM vendor B</td>
<td>yes</td>
</tr>
<tr>
<td>Model #5</td>
<td>2013</td>
<td>Family X</td>
<td>DRAM vendor B</td>
<td>yes</td>
</tr>
<tr>
<td>Model #5</td>
<td>2013</td>
<td>Family X</td>
<td>DRAM vendor B</td>
<td>yes</td>
</tr>
<tr>
<td>Model #5</td>
<td>2013</td>
<td>Family X</td>
<td>DRAM vendor B</td>
<td>yes</td>
</tr>
<tr>
<td>Model #5</td>
<td>2013</td>
<td>Family X</td>
<td>DRAM vendor B</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>Laptop model</td>
<td>Laptop year</td>
<td>CPU family (microarchitecture)</td>
<td>DRAM manufacturer</td>
</tr>
<tr>
<td>---</td>
<td>--------------</td>
<td>-------------</td>
<td>---------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>16</td>
<td>Model #5</td>
<td>2013</td>
<td>Family X</td>
<td>DRAM vendor B</td>
</tr>
<tr>
<td>17</td>
<td>Model #5</td>
<td>2013</td>
<td>Family X</td>
<td>DRAM vendor C</td>
</tr>
<tr>
<td>18</td>
<td>Model #5</td>
<td>2013</td>
<td>Family X</td>
<td>DRAM vendor C</td>
</tr>
<tr>
<td>19</td>
<td>Model #5</td>
<td>2013</td>
<td>Family X</td>
<td>DRAM vendor C</td>
</tr>
<tr>
<td>20</td>
<td>Model #5</td>
<td>2013</td>
<td>Family X</td>
<td>DRAM vendor C</td>
</tr>
<tr>
<td>21</td>
<td>Model #5</td>
<td>2013</td>
<td>Family X</td>
<td>DRAM vendor C</td>
</tr>
<tr>
<td>22</td>
<td>Model #5</td>
<td>2013</td>
<td>Family X</td>
<td>DRAM vendor C</td>
</tr>
<tr>
<td>23</td>
<td>Model #6</td>
<td>2013</td>
<td>Family Y</td>
<td>DRAM vendor A</td>
</tr>
<tr>
<td>24</td>
<td>Model #6</td>
<td>2013</td>
<td>Family Y</td>
<td>DRAM vendor B</td>
</tr>
<tr>
<td>25</td>
<td>Model #6</td>
<td>2013</td>
<td>Family Y</td>
<td>DRAM vendor B</td>
</tr>
<tr>
<td>26</td>
<td>Model #6</td>
<td>2013</td>
<td>Family Y</td>
<td>DRAM vendor B</td>
</tr>
<tr>
<td>27</td>
<td>Model #6</td>
<td>2013</td>
<td>Family Y</td>
<td>DRAM vendor B</td>
</tr>
<tr>
<td>28</td>
<td>Model #7</td>
<td>2012</td>
<td>Family W</td>
<td>DRAM vendor D</td>
</tr>
<tr>
<td>29</td>
<td>Model #8</td>
<td>2014</td>
<td>Family Z</td>
<td>DRAM vendor A</td>
</tr>
</tbody>
</table>

15/29 Machines were vulnerable...
06 Rowhammer defenses
Rowhammer detection

- Software binary analysis
Rowhammer detection

- Software binary analysis

Rowhammer neutralization

- *G-CATT
  
  ✓ Isolate user space / kernel space in physical memory
  
  ✓ Attacker cannot exploit bit flips in kernel memory

*“CAn’t Touch This: Software-only Mitigation against Rowhammer Attacks targeting Kernel Memory”, F.Brasser et al. (2017.08)
Rowhammer detection

- Software binary analysis

Rowhammer neutralization

- \*G-CATT
  - ✓ Isolate user space / kernel space in physical memory
  - ✓ Attacker cannot exploit bit flips in kernel memory

Rowhammer elimination

- TRR (Target Row Refresh) : Identify frequently accessed DRAM addresses
- tREFI (time of REfresh Interval)  ➔ e.g. Intel Skylake, Kaby lake
- ECC memory (Error Correcting Code)

* “Can’t Touch This: Software-only Mitigation against Rowhammer Attacks targeting Kernel Memory”, F.Brasser et al. (2017.08)
Mark Lanteigne, Third I/O CTO and founder, told Ars there's no immediate danger of Rowhammer being exploited maliciously to hijack the security of computers that use the vulnerable memory chips. Still, he said his assessment presents a significantly less comforting picture than those painted by Samsung, Micron, and other DDR manufacturers. Samsung, he said, has largely declared its DDR4 product line to be "Rowhammer free" because of technology it calls TRR, or targeted row refresh, which makes chips better able to withstand large numbers of malicious accesses that come in rapid succession during the attack. Micron, meanwhile, has also praised the benefits of TRR in its DDR4 products.

**DDR4 and Rowhammer**

When Rowhammer was first discovered and discussed, Samsung claimed that its DDR4 would not be susceptible to this attack method due to its use of Targeted Row Refresh inside devices. Micron followed suit with a statement that TRR mode is implemented in the background of its hardware as well. Third I/O's testing shows that in Micron's case, at least, this protection is imperfectly implemented. The paper states:

- TRR (Target Row Refresh) : Identify frequently accessed DRAM addresses
- tREFI (time of REFresh Interval) ➔ e.g. Intel Skylake, Kaby lake
- ECC memory (Error Correcting Code)

** https://arstechnica.com/information-technology/2016/03/once-thought-safe-ddr4-memory-shown-to-be-vulnerable-to-rowhammer/
Conclusion & Recent study
Rowhammer attack on flash memory
- IBM

Another Flip in the Wall of Rowhammer Defenses
- Daniel Gruss et al.

Exploiting the DRAM rowhammer bug to gain kernel privileges
- Google project zero

Flipping Bits in Memory Without Accessing Them
- Yoongu Kim (CMU) et al.

(2014.07)
(2015.03)
(2017.08)
(2017.10)
Rowhammer attack on flash memory
- IBM

Another Flip in the Wall of Rowhammer Defenses
- Daniel Gruss et al.

Exploiting the DRAM rowhammer bug to gain kernel privileges
- Google project zero

Flipping Bits in Memory Without Accessing Them
- Yoongu Kim (CMU) et al.
Rowhammer attack on flash memory
- IBM

Exploiting the DRAM rowhammer bug to gain kernel privileges
- Google project zero

Flipping Bits in Memory Without Accessing Them
- Yoongu Kim (CMU) et al.

Another Flip in the Wall of Rowhammer Defenses
- Daniel Gruss et al.
Flipping Bits in Memory Without Accessing Them
- Yoongu Kim (CMU) et al. (2014.07)

Rowhammer attack on flash memory
- IBM (2015.03)

Exploiting the DRAM rowhammer bug to gain kernel privileges
- Google project zero (2017.08)

Another Flip in the Wall of Rowhammer Defenses
- Daniel Gruss et al. (2017.10)
Another Flip in the Wall of Rowhammer Defenses - Daniel Gruss et al.

▲ Ordinary rowhammer

▲ One-location hammering
Future work
It might be a good mitigation...

- Arrange Refresh-only row buffer
It might be a good mitigation...

- Arrange Refresh-only row buffer
It might be a good mitigation...

- Arrange Refresh-only row buffer
It might be a good mitigation...

- Arrange Refresh-only row buffer

Refresh-only row buffer

▲ Ordinary rowhammer

▲ One-location hammering
Q/A
THANK YOU