Drammer: Deterministic RowHammer Attacks on Mobile Platforms

Victor van der Veen       Daniel Gruss       Herbert Bos
Yanick Fratantonio       Clémentine Maurice   Kaveh Razavi
Martina Lindorfer        Giovanni Vigna       Cristiano Giuffrida

Vrije Universiteit Amsterdam, UC Santa Barbara, Graz University of Technology

Association for Computing Machinery Conference on Computer and Communication Security (ACM CCS) 2016

Presented by: Lin Xiao
Summary

- **Motivation**
  - Previous works on RowHammer attacks either are *only probabilistic attacks* (weak reliability) or rely on special memory management features
  - RowHammer demonstrated only on x86
    - Unclear whether RowHammer attack is feasible on mobile platforms
- **Goal**
  - Prove RowHammer is possible on ARM/Android
  - Implement the first deterministic RowHammer based Android root exploit
    - Which solely relies on *predictable memory reuse pattern* of standard physical memory allocators
    - Relies on *no software vulnerability* and requires *no user permission*
- **Idea**
  - Phys Feng Shui
    - Abuses predictable memory reuse pattern of the physical memory allocators
  - Uses this technique to implement an Android root exploit: DRAMMER
- **Result**
  - Many of the popular smartphones are susceptible to Drammer attack
Outline

1. RowHammer Review
2. Goal
3. Preliminary Feasibility Analysis
4. Exploitation Primitives
5. DRAMMER Attack
6. Results
7. Existing RowHammer Defences & Countermeasure Against Drammer
8. Strengths & Weaknesses
9. Discussion
RowHammer review

- RowHammer allows an attacker to modify memory without accessing it, simply by hammering the adjacent rows of the physical memory location in the DRAM chip until a bit flips.

![DRAM Rows Diagram]

- DRAM Rows
- Aggressor row
- Victim row
- Aggressor row
Goal

- Prove that RowHammer is possible on ARM/Android
  - Prior works demonstrated RowHammer attacks only on x86 systems

- Implement the first deterministic RowHammer based Android root exploit
  - Which solely relies on predictable memory reuse pattern of standard physical memory allocators
  - Relies on no software vulnerability and requires no root permission
Preliminary Feasibility Analysis (RowhARMer)

- Device used: LG Nexus 5 with Android 6.0.1
- Flips occurred in a matter of seconds
  - Up to 150 flips per minute
- Access times of 300 ns or higher are unlikely to trigger bit flips
- “sweet spot” for triggering the most flips on this particular DRAM chip was not at full speed but at 100ns per read

It is possible to trigger bit flips in seconds and up to 150 flips per minutes can be triggered at about 100ns per read
Outline

1. RowHammer Review
2. Goal
3. Preliminary Feasibility Analysis
4. Exploitation Primitives
   1. Exploitation on x86 Architecture
   2. Additional Challenges
5. DRAMMER Attack
6. Results
7. Existing RowHammer Defences & Countermeasure Against Drammer
8. Strengths & Weaknesses
9. Discussion
Exploitation Primitives

- **Fast Uncached Memory Access**
  - The ability of activating rows in each bank fast enough → bypass caches

- **Physical Memory Massaging**
  - Massage the memory precisely to push the victim to use the vulnerable cell to store security sensitive data

- **Physical Memory Addressing**
  - Mapping from virtual space to physical memory
  - Crucial for double-sided RowHammer
Exploitation on x86 Architecture – P1 Fast Uncached Memory Access

- **Explicit Cache Flush**
  - clflush instruction: flushes the cache entry associated with the given address
  - On ARMv7, the cache flush instruction is privileged, therefore only executable on kernel
    - Android kernel exposes cacheflush() instruction, but only flushes up to level 2 cache
  - ARMv8 provide unprivileged cache flush instructions, may be disabled by the kernel

- **Cache Eviction Sets:**
  - Repeatedly accessing memory addresses which belong to the same cache eviction set (set of conguerent (map to the same cache line) addresses)
  - Proved to be too slow in practice to trigger bit flips on both ARMv7, ARMv8

- **Non-temporal Access Instructions:**
  - Non-temporal: the data will not likely be reused soon → does not have to be cached
  - ARMv8 offers non-temporal load and store, but only serves as a hint to CPU. In practice that memory still remains cached
Exploitation on x86 Architecture – P2 Physical Memory Massaging

- **Page-table Spraying**
  - Probabilistic exploitation strategy
    - Sprays the memory with page table, with the hope that at least one of the bits lands on a vulnerable physical memory page

- **Memory Deduplication:**
  - Trick the OS into mapping two pages
    - An attacker-controlled virtual memory page and a victim-owned virtual memory page (not always on)
  - Perform RowHammer on attacker-controlled page and physical memory leading to access in kernel memory

- **MMU Paravirtualization**
  - Allows a guest VM to specify the physical location of a (read-only) PTP
  - Allowing a malicious VM to trick the VM monitor into mapping a page table into a vulnerable location to hammer (not always on)
Exploitation on x86 Architecture – P3 Physical Memory Addressing

- **Pagemap Interface**
  - `/proc/self/pagemap` file
    - Contains complete information about mapping of virtual to physical addresses
    - Unprivileged access no longer allowed on the Android Linux kernel

- **Huge Virtual Pages**
  - Backed by physically contiguous physical pages
  - Covers 2MB of contiguous addresses
    - Not as fine-grained as knowing absolute physical addresses
    - Uses relative offsets to access specific physical memory pages for double-sided RowHammer
    - Guarantees that two rows that are contiguous in virtual memory are also contiguous in physical memory
    - (Stock Android disabled by default and does not exist for every mobile device)
### Summary

<table>
<thead>
<tr>
<th>Primitive</th>
<th>x86 Platforms</th>
<th>Mobile Devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast Uncached Memory Access</td>
<td></td>
<td>ARMv7/ARMv8</td>
</tr>
<tr>
<td>Explicit cache flush</td>
<td>●</td>
<td>○/○</td>
</tr>
<tr>
<td>Cache eviction sets</td>
<td>●</td>
<td>—/—</td>
</tr>
<tr>
<td>Non-temporal access instructions</td>
<td>●</td>
<td>—/○</td>
</tr>
<tr>
<td>Physical Memory Massaging</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Page-table spraying</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>Memory deduplication</td>
<td>●</td>
<td>—</td>
</tr>
<tr>
<td>MMU paravirtualization</td>
<td>●</td>
<td>—</td>
</tr>
<tr>
<td>Physical Memory Addressing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pagemap interface</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>Huge pages</td>
<td>●</td>
<td>—</td>
</tr>
</tbody>
</table>

- ●: unprivileged mode
- ○: privileged mode
- ○: not practical enough to use in our setting
- —: does not exist

Exploitation of x86 cannot be used for ARM
Additional Challenges

- ARM specifications do not provide memory details
  - Not clear what the size of a row is

- Mobile devices do not have any swap space
  - The OS – the Low Memory Killer in particular on Android – starts killing processes if the memory pressure is too high
Outline

1. RowHammer Review
2. Goal
3. Preliminary Feasibility Analysis
4. Exploitation Primitives
5. DRAMMER Attack
   1. Mobile Device Memory
      1. DMA
      2. Buddy Allocator
   2. Phys Feng Shui
3. Implementation on Android
6. Results
7. Existing RowHammer Defences & Countermeasure Against Drammer
8. Strengths & Weaknesses
9. Discussion

14
DRAMMER
Mobile Device Memory

- ARM neither document the row size nor provide its platform instructions for fingerprinting DRAM modules
- Relies on observation that accessing two memory pages from the same bank is slower than reading from different banks
Direct Memory Access (DMA)

- Used to support efficient memory sharing between hardware components as well as between devices and userlands

- Provide P1 (fast access) and P3 (mapping) of the attack primitives
  - Bypasses CPU and its caches
  - Most devices perform DMA operations only to physically contiguous memory pages
    - The OS provides allocators that support this type of memory

- On Android: ION memory allocator
Buddy Allocator

<table>
<thead>
<tr>
<th>Physical Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>4KB</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

16*4KB pages = 64KB rows
Physical Memory

Allocate: 16 KB

16*4KB pages = 64 KB rows
Buddy Allocator

Physical Memory

Allocate: 16 KB

16*4KB pages = 64 KB rows
Buddy Allocator

Physical Memory

Allocate: 16 KB

16*4KB pages = 64 KB rows
Buddy Allocator

Physical Memory

Allocate: 16 KB

16*4KB pages = 64 KB rows
Buddy Allocator

Physical Memory

Allocate: 16 KB

16*4KB pages = 64 KB rows
Buddy Allocator

Physical Memory

Allocate: 8 KB

16*4KB pages = 64 KB rows
Physical Memory

Allocate: 64 KB

16*4KB pages = 64 KB rows
Buddy Allocator

Physical Memory

Free: 8 KB

16*4KB pages = 64 KB rows
Buddy Allocator

Physical Memory

Free: 8 KB

16*4KB pages = 64 KB rows
Phys Feng Shui

- Used for physical memory massaging
- Generic technique for deterministic RowHammer exploitation using commodity features
Phys Feng Shui

![Diagram of memory allocation]

- **L** represents allocated memory.
- **M** represents free memory.
- **S** represents uninitialized memory.

**UNINITIALIZED MEMORY**
Exploitation Templates

- Determined by a combination of the number of flips found in potential PTEs and the relative location of each flip in target templates

- For ARMv7 with maximum L size of 4MB template is not exploitable if
  - It falls in the second half of a page (a shadow page)
  - It falls in the lowest 12 bits of 32-bit word (properties field of a PTE)
  - It falls in the highest 11 bits of a 32 bit word (bit flip)

- Therefore at most 9 bits are exploitable and since there are only 256 hardware PTEs per pages

at most 7.0% out of all possible bits of a single page are exploitable
All available physically contiguous chunks of size $L$ are exhausted
Probed for vulnerable template
Phys Feng Shui

- **All chunks of size $M$ is exhausted**
- **Blocks of size $M$ and larger are no longer available**
- Select target for exploit -> corresponding L block = L*
- L* released
- All M chunks exhausted
- M chunk that holds exploitable template -> M*
  - Choice restricted to chunks not at the edge of L*
Phys Feng Shui

- M* is released
- Remaining L chunks freed to avoid system crash
Allocating $S$ chunks till subsequent $S$ chunk lands in $M^*$
Allocation of a number of padding pages
Map the PTE with a bit flip at offset bit $n$ to a location $2^n$ pages away from the PT.

The bit-flip in PT will cause the page table entry to point to PT itself.
Exploitation

- Double-sided RowHammer → replicating the bit flip found in template phase
Exploitation

- After bit flip PT is now mapped to our address space
- Fill PT with PTE’s to kernel memory
- Search for the **security context** of our own process stored in a `struct cred`
- Overwrite our **uid** and **gid** to get root privileges

Once flip triggered, write access gained to the PT, therefore access gained to any page in physical memory including kernel memory
Implementation on Android

- Android provides DMA Buffer Management APIs through its main memory manager called ION
  - Allows userland apps to access uncached, physically contiguous memory

- Noise elimination
  - Ensure reliability
    - eliminate interferences from other activity in the system during Phys Feng Shui phase
  - Risk of interferences is minimal because Phys Feng Shui is extremely short lived
  - Interferences are only possible when the kernel independently allocates memory via the buddy allocator in the low memory zone
  - Minimize the risk of noise by scheduling the attack during low system activity (ACTION_SCREEN_OFF or ACTION_BATTERY_LOW)
Outline

1. RowHammer Review
2. Goal
3. Preliminary Feasibility Analysis
4. Exploitation Primitives
5. DRAMMER Attack
6. Results
   1. Methodology
   2. Analysis
7. Existing RowHammer Defences & Countermeasure Against Drammer
8. Strengths & Weaknesses
9. Discussion
Methodology

- **Architecture:**
  - ARMv7
  - ARMv8

- **DRAM Types:**
  - LPDDR2/3/4

- **Metrics:**
  - Time until first bit-flip
  - Number of bit-flips
  - Number exploitable bit-flips
81% of ARMv7 devices vulnerable and 16% of ARMv8 devices vulnerable
### Results

### ARMv7

<table>
<thead>
<tr>
<th>Device</th>
<th>Hardware Details</th>
<th>Analysis Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nexus 5_1</td>
<td>MSM8974&lt;sup&gt;1&lt;/sup&gt; 2 GB 64</td>
<td>MB</td>
</tr>
<tr>
<td>Nexus 5_2</td>
<td>MSM8974&lt;sup&gt;1&lt;/sup&gt; 2 GB 64</td>
<td>441</td>
</tr>
<tr>
<td>Nexus 5_3</td>
<td>MSM8974&lt;sup&gt;1&lt;/sup&gt; 2 GB 64</td>
<td>472</td>
</tr>
<tr>
<td>Nexus 5_4</td>
<td>MSM8974&lt;sup&gt;1&lt;/sup&gt; 2 GB 64</td>
<td>461</td>
</tr>
<tr>
<td>Nexus 5_5</td>
<td>MSM8974&lt;sup&gt;1&lt;/sup&gt; 2 GB 64</td>
<td>616</td>
</tr>
<tr>
<td>Nexus 5_6</td>
<td>MSM8974&lt;sup&gt;1&lt;/sup&gt; 2 GB 64</td>
<td>630</td>
</tr>
<tr>
<td>Nexus 5_7</td>
<td>MSM8974&lt;sup&gt;1&lt;/sup&gt; 2 GB 64</td>
<td>512</td>
</tr>
<tr>
<td>Nexus 5_8</td>
<td>MSM8974&lt;sup&gt;1&lt;/sup&gt; 2 GB 64</td>
<td>485</td>
</tr>
<tr>
<td>Nexus 5_9</td>
<td>MSM8974&lt;sup&gt;1&lt;/sup&gt; 2 GB 64</td>
<td>569</td>
</tr>
<tr>
<td>Nexus 5_10</td>
<td>MSM8974&lt;sup&gt;1&lt;/sup&gt; 2 GB 64</td>
<td>406</td>
</tr>
<tr>
<td>Nexus 5_11</td>
<td>MSM8974&lt;sup&gt;1&lt;/sup&gt; 2 GB 64</td>
<td>613</td>
</tr>
<tr>
<td>Nexus 5_12</td>
<td>MSM8974&lt;sup&gt;1&lt;/sup&gt; 2 GB 64</td>
<td>600</td>
</tr>
<tr>
<td>Nexus 5_13</td>
<td>MSM8974&lt;sup&gt;1&lt;/sup&gt; 2 GB 64</td>
<td>579</td>
</tr>
<tr>
<td>Nexus 5_14</td>
<td>MSM8974&lt;sup&gt;1&lt;/sup&gt; 2 GB 64</td>
<td>576</td>
</tr>
<tr>
<td>Nexus 5_15</td>
<td>MSM8974&lt;sup&gt;1&lt;/sup&gt; 2 GB 64</td>
<td>573</td>
</tr>
<tr>
<td>Nexus 5_17</td>
<td>MSM8974&lt;sup&gt;1&lt;/sup&gt; 2 GB 64</td>
<td>621</td>
</tr>
<tr>
<td>Galaxy S5</td>
<td>MSM8974&lt;sup&gt;1&lt;/sup&gt; 2 GB 64</td>
<td>207</td>
</tr>
<tr>
<td>OnePlus One&lt;sub&gt;1&lt;/sub&gt;</td>
<td>MSM8974&lt;sup&gt;1&lt;/sup&gt; 3 GB 64</td>
<td>292</td>
</tr>
<tr>
<td>OnePlus One&lt;sub&gt;2&lt;/sub&gt;</td>
<td>MSM8974&lt;sup&gt;1&lt;/sup&gt; 3 GB 64</td>
<td>1,189</td>
</tr>
<tr>
<td>Moto G&lt;sub&gt;2013&lt;/sub&gt;</td>
<td>MSM8226 1 GB 32</td>
<td>134</td>
</tr>
<tr>
<td>Moto G&lt;sub&gt;2014&lt;/sub&gt;</td>
<td>MSM8226 1 GB 32</td>
<td>151</td>
</tr>
<tr>
<td>Nexus 4</td>
<td>APQ064 2 GB&lt;sup&gt;2&lt;/sup&gt; 64</td>
<td>82</td>
</tr>
</tbody>
</table>

### ARMv8

<table>
<thead>
<tr>
<th>Device</th>
<th>Hardware Details</th>
<th>Analysis Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nexus 5x</td>
<td>MSM8992 2 GB 64</td>
<td>MB</td>
</tr>
<tr>
<td>Galaxy S6</td>
<td>Exynos7420 3 GB 128</td>
<td>271</td>
</tr>
<tr>
<td>K3 Note</td>
<td>MT6752 2 GB 64</td>
<td>234</td>
</tr>
<tr>
<td>Mi 4i</td>
<td>MSM8992 2 GB 64</td>
<td>327</td>
</tr>
<tr>
<td>Desire 510</td>
<td>MSM8916 1 GB 32</td>
<td>186</td>
</tr>
<tr>
<td>G4</td>
<td>MSM8992 3 GB 64</td>
<td>833</td>
</tr>
</tbody>
</table>

<sup>1</sup> MSM8974A<sup>1</sup> 2<sup>sup</sup> LPDDR2<sup>4</sup> 3<sup>sup</sup> LPDDR4

Same modell can be **both vulnerable and not vulnerable**
Results

Time until the first flip can vary and number of bit flips can vary

<table>
<thead>
<tr>
<th>Device</th>
<th>Hardware Details</th>
<th>Analysis Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SoC</td>
<td>DRAM</td>
</tr>
<tr>
<td>Nexus 5k</td>
<td>MSM8974</td>
<td>2 GB</td>
</tr>
<tr>
<td>Nexus 5</td>
<td>MSM8974</td>
<td>2 GB</td>
</tr>
<tr>
<td>Nexus 5</td>
<td>MSM8974</td>
<td>2 GB</td>
</tr>
<tr>
<td>Nexus 5</td>
<td>MSM8974</td>
<td>2 GB</td>
</tr>
<tr>
<td>Nexus 5</td>
<td>MSM8974</td>
<td>2 GB</td>
</tr>
<tr>
<td>Nexus 5</td>
<td>MSM8974</td>
<td>2 GB</td>
</tr>
<tr>
<td>Nexus 5</td>
<td>MSM8974</td>
<td>2 GB</td>
</tr>
<tr>
<td>Nexus 5</td>
<td>MSM8974</td>
<td>2 GB</td>
</tr>
<tr>
<td>Nexus 5</td>
<td>MSM8974</td>
<td>2 GB</td>
</tr>
<tr>
<td>Nexus 5</td>
<td>MSM8974</td>
<td>2 GB</td>
</tr>
<tr>
<td>Nexus 5</td>
<td>MSM8974</td>
<td>2 GB</td>
</tr>
<tr>
<td>Nexus 5</td>
<td>MSM8974</td>
<td>2 GB</td>
</tr>
<tr>
<td>Nexus 5</td>
<td>MSM8974</td>
<td>2 GB</td>
</tr>
<tr>
<td>Nexus 5</td>
<td>MSM8974</td>
<td>2 GB</td>
</tr>
<tr>
<td>Galaxy S5</td>
<td>MSM8974</td>
<td>2 GB</td>
</tr>
<tr>
<td>OnePlus One</td>
<td>MSM8974</td>
<td>3 GB</td>
</tr>
<tr>
<td>OnePlus One</td>
<td>MSM8974</td>
<td>3 GB</td>
</tr>
<tr>
<td>Moto G2013</td>
<td>MSM8226</td>
<td>1 GB</td>
</tr>
<tr>
<td>Moto G2014</td>
<td>MSM8226</td>
<td>1 GB</td>
</tr>
<tr>
<td>Nexus 4</td>
<td>APQ8064</td>
<td>2 GB</td>
</tr>
</tbody>
</table>

1 MSM8974AA 2 MSM8974AC 3 LPDDR2 4 LPDDR4
<table>
<thead>
<tr>
<th>Device</th>
<th>Hardware Details</th>
<th>Analysis Results</th>
<th>Analysis Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SoC</td>
<td>DRAM</td>
<td>RS</td>
</tr>
<tr>
<td>Nexus 5i</td>
<td>MSM8974#1</td>
<td>2 GB</td>
<td>64</td>
</tr>
<tr>
<td>Nexus 5r</td>
<td>MSM8974#1</td>
<td>2 GB</td>
<td>64</td>
</tr>
<tr>
<td>Nexus 5s</td>
<td>MSM8974#1</td>
<td>2 GB</td>
<td>64</td>
</tr>
<tr>
<td>Nexus 5x</td>
<td>MSM8974#1</td>
<td>2 GB</td>
<td>64</td>
</tr>
<tr>
<td>Nexus 5i5</td>
<td>MSM8974#1</td>
<td>2 GB</td>
<td>64</td>
</tr>
<tr>
<td>Nexus 5i6</td>
<td>MSM8974#1</td>
<td>2 GB</td>
<td>64</td>
</tr>
<tr>
<td>Nexus 5t</td>
<td>MSM8974#1</td>
<td>2 GB</td>
<td>64</td>
</tr>
<tr>
<td>Nexus 5g</td>
<td>MSM8974#1</td>
<td>2 GB</td>
<td>64</td>
</tr>
<tr>
<td>Nexus 5i9</td>
<td>MSM8974#1</td>
<td>2 GB</td>
<td>64</td>
</tr>
<tr>
<td>Nexus 5i10</td>
<td>MSM8974#1</td>
<td>2 GB</td>
<td>64</td>
</tr>
</tbody>
</table>

**ARMv7**

<table>
<thead>
<tr>
<th>Device</th>
<th>Hardware Details</th>
<th>Analysis Results</th>
<th>Analysis Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nexus 5i2</td>
<td>MSM8974#1</td>
<td>2 GB</td>
<td>64</td>
</tr>
<tr>
<td>Nexus 5i3</td>
<td>MSM8974#1</td>
<td>2 GB</td>
<td>64</td>
</tr>
<tr>
<td>Nexus 5i4</td>
<td>MSM8974#1</td>
<td>2 GB</td>
<td>64</td>
</tr>
<tr>
<td>Nexus 5i5</td>
<td>MSM8974#1</td>
<td>2 GB</td>
<td>64</td>
</tr>
<tr>
<td>Nexus 5i6</td>
<td>MSM8974#1</td>
<td>2 GB</td>
<td>64</td>
</tr>
<tr>
<td>Galaxy S5</td>
<td>MSM8974#2</td>
<td>2 GB</td>
<td>64</td>
</tr>
<tr>
<td>OnePlus One51</td>
<td>MSM8974#2</td>
<td>3 GB</td>
<td>64</td>
</tr>
<tr>
<td>OnePlus One52</td>
<td>MSM8974#3</td>
<td>3 GB</td>
<td>64</td>
</tr>
<tr>
<td>Moto G2013</td>
<td>MSM8226#1</td>
<td>1 GB</td>
<td>32</td>
</tr>
<tr>
<td>Moto G2014</td>
<td>MSM8226#2</td>
<td>1 GB</td>
<td>32</td>
</tr>
<tr>
<td>Nexus 4</td>
<td>APQ8064#2</td>
<td>2 GB</td>
<td>64</td>
</tr>
</tbody>
</table>

**ARMv8**

<table>
<thead>
<tr>
<th>Device</th>
<th>Hardware Details</th>
<th>Analysis Results</th>
<th>Analysis Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nexus 5x</td>
<td>MSM8902#1</td>
<td>2 GB</td>
<td>64</td>
</tr>
<tr>
<td>Galaxy S6</td>
<td>Exynos7420#3</td>
<td>3 GB</td>
<td>128</td>
</tr>
<tr>
<td>K3 Note</td>
<td>MT6732#4</td>
<td>2 GB</td>
<td>64</td>
</tr>
<tr>
<td>Mi 4i</td>
<td>MSM8693#4</td>
<td>2 GB</td>
<td>64</td>
</tr>
<tr>
<td>Desire 510</td>
<td>MSM8916#4</td>
<td>1 GB</td>
<td>32</td>
</tr>
<tr>
<td>G4</td>
<td>MSM8692#4</td>
<td>2 GB</td>
<td>64</td>
</tr>
</tbody>
</table>

1 MSM979AA 2 MSM979AC 3 LPDDR2 4 LPDDR4

LPDDR2/3 is vulnerable and LPDDR4 maybe vulnerable
Existing RowHammer Defenses

- **Software based**
  - Instruction «blacklisting»: disallowing to rewriting instruction such as clfluss and non temporal instructions
    - Linux pagemap interface is now prohibited from userland
  - Cache monitoring
    - ANVIL
    - Herath et al

- **Hardware based**
  - Error Correcting Codes
    - Correct single bit flips errors and reports other errors
    - Some vendors implement ECC to report bit flip error only upon reaching a certain threshold
  - Doubling DRAM refresh rates
    - Severely limits most attacks
  - ARMOR
  - PARA

Drammer bypasses the cache completely through DMA and neither use special instr. nor page map interface, so it can bypass the Software mitigations
Countermeasures Against DRAMMER

- **Restriction of userland interface**
  - adopt constraint-based allocation

- **Memory isolation and integrity**
  - Stricter enforcement of memory isolation
  - Completely isolate DMA-able memory from other regions
    - Isolate ION regions controlled by userland from kernel memory
  - Design of isolation and integrity measures for security critical data such as pages tables also needs improvements

- **Prevention of memory exhaustion**
  - Per process memory limits would make it harder for an attacker
    - To find exploitable templates
    - To exhaust all available memory chunks of different sizes during Phys Feng Shui
  - Enforce this limit at the OS level (both for user and kernel memory) and per user ID (to prohibit collusion)
Strengths

- First of its kind to implement RowHammer on mobile platform and ARM architecture
- Attack is deterministic and solely rely on predictable memory reuse pattern of standard physical memory allocators
- Implementation discussed in detail
Weaknesses

- Only works for the buddy allocator

- Unknown whether it works on other OS other than Android
  - Generalization aspect is discussed, but it does not cover all aspects

- Only discussed for Linux-based platforms
Related Work

- **Flip Feng Shui: Hammering a Needle in the Software Stack**
  - Kaveh Razavi, Ben Gras, and Erik Bosman, Bart Preneel, Cristiano Giuffrida and Herbert Bos
  - [sec16_paper_razavi.pdf](https://usenix.org) (usenix.org)

- **TRRespass: Exploiting the Many Sides of Target Row Refresh**
  - Pietro Frigo, Emanuele Vannacci, Hasan Hassan, Victor van der Veen, Onur Mutlu, Cristiano Giuffrida, Herbert Bos, Kaveh Razavi
  - [rowhammer-TRRespass_ieee_security_privacy20.pdf](https://ethz.ch) (ethz.ch)

- **Introducing Half-Double: New Hammering Technique for DRAM RowHammer Bug**
  - Salman Qazi, Yoongu Kim, Nicolas Boichat, Eric Shiu & Mattias Nissler
Discussion && Question?
Discussion starters

- Thoughts on the proposed ideas to exploit RowHammer
- Thoughts on additional mitigation for this attack
- Can this attack still function on the devices of today?
- Can this be applied to iOS or any other systems?
- Are other mobile devices (other than smart phones) susceptible to this attack?