Drammer: Deterministic Rowhammer Attacks on Mobile Platforms

by

A bunch of pasty faced sad sack nerds sitting in a basement want to sound cool and tough, like they've just done a tour in 'Nam. [slashdot]
Drammer:
Deterministic Rowhammer Attacks
on Mobile Platforms

Victor van der Veen\textsuperscript{1}, Yanick Fratantonio\textsuperscript{2}, Martina Lindorfer\textsuperscript{2}, Daniel Gruss\textsuperscript{3}, Clémentine Maurice\textsuperscript{3}, Giovanni Vigna\textsuperscript{2}, Herbert Bos\textsuperscript{1}, Kaveh Razavi\textsuperscript{1}, and Cristiano Giuffrida\textsuperscript{1}

\textsuperscript{1}Vrije Universiteit Amsterdam, \textsuperscript{2}UC Santa Barbara, \textsuperscript{3}TU Graz
Drammer: Deterministic Rowhammer Attacks on Mobile Platforms

Your takeaway message of today
Drammer: Deterministic Rowhammer Attacks on Mobile Platforms

Your takeaway message of today

Rowhammer on ARM
Drammer: Deterministic Rowhammer Attacks on Mobile Platforms

Your takeaway message of today

Rowhammer on ARM

Deterministic exploitation
Drammer: Deterministic Rowhammer Attacks on Mobile Platforms

Your takeaway message of today

Rowhammer on ARM
Deterministic exploitation
Works on a Google Pixel
Drammer: Deterministic Rowhammer Attacks on Mobile Platforms

Flipping bits in memory

DRAM hardware glitch causing disturbance errors

```
    1 1 0 1 1
    1 0 1 1 0
    0 1 1 0 0
    1 1 0 1 0
    0 0 1 0 1
```

Aggressor row
Victim row
Aggressor row
Drammer: Deterministic Rowhammer Attacks on Mobile Platforms

Flipping bits in memory

DRAM hardware glitch causing disturbance errors

1 1 0 1 1 1
1 0 1 1 1 0
0 1 1 0 0 0
1 1 0 1 0 0
1 1 0 1 0 0
0 0 1 0 1 1

Aggressor row
Victim row
Aggressor row
Rowhammer: Deterministic Rowhammer Attacks on Mobile Platforms

Flipping bits in memory

DRAM hardware glitch causing disturbance errors

1 1 1 1
1 0 1 1
0 1 1 0
1 1 0 1
0 0 1 0

Aggressor row
Victim row
Aggressor row
Drammer: Deterministic Rowhammer Attacks on Mobile Platforms

Flipping bits in memory

DRAM hardware glitch causing disturbance errors

```
1 1 0 1 1 1
1 0 1 1 1 0
0 1 1 1 0 0
1 1 0 1 0 0
1 1 0 1 0 0
0 0 1 0 1 1
```
Drammer: Deterministic Rowhammer Attacks on Mobile Platforms

Flipping bits in memory

DRAM hardware glitch causing disturbance errors

![Diagram of DRAM memory rows with highlighted aggressor and victim rows]
Drammer: Deterministic Rowhammer Attacks on Mobile Platforms

Flipping bits in memory

DRAM hardware glitch causing disturbance errors

![Diagram of memory rows]

1 1 0 1 1 1
1 0 1 1 1 0
0 1 1 1 0 0
1 1 0 1 1 0
0 0 1 0 1 1

Aggressor row
Victim row
Aggressor row
Drammer: Deterministic Rowhammer Attacks on Mobile Platforms

Flipping bits in memory

DRAM hardware glitch causing disturbance errors

Aggressor row
Victim row
Aggressor row
Drammer: Deterministic Rowhammer Attacks on Mobile Platforms

Flipping bits in memory

DRAM hardware glitch causing disturbance errors

Diagram of aggressor and victim rows with binary values highlighting the disturbance.
Drammer: Deterministic Rowhammer Attacks on Mobile Platforms

Flipping bits in memory

DRAM hardware glitch causing disturbance errors

- Not every bit may flip
- Once a bit flips, we can reproduce it
1. **Memory Templating**
   Scan memory for useful bit flips

Overview
Overview

1. Memory Templating
   Scan memory for useful bit flips
Overview

1. **Memory Templating**
   Scan memory for useful bit flips

2. **Land sensitive data**
   Store a crucial data structure on a vulnerable page
1. **Memory Templating**  
   Scan memory for useful bit flips

2. **Land sensitive data**  
   Store a crucial data structure on a vulnerable page

3. **Reproduce the bit flip**  
   Modify the data structure and get root access
1. Memory Templating
   Scan memory for useful bit flips
Uncached memory access
- clflush
- cache eviction
- non-temporal access instructions

Determining the physical addresses aggressor/victim rows
- /proc/self/pagemap
- 2MB huge pages (relative)
Uncached memory access
- `clflush`
- cache eviction
- non-temporal access instructions

Determining the physical addresses aggressor/victim rows
- `/proc/self/pagemap`
- 2MB huge pages (relative)

But does it work on ARM?
Templating

Uncached memory access
• clflush
• cache eviction
• non-temporal access instructions

Determining the physical addresses aggressor/victim rows
• /proc/self/pagemap
• 2MB huge pages (relative)

But does it work on ARM?

Nope
Uncached memory access
- clflush
- cache eviction
- non-temporal access instructions

Determining the physical addresses aggressor/victim rows
- /proc/self/pagemap
- 2MB huge pages (relative)

But does it work on ARM?

None of them
Uncached memory access
• `clflush`
• cache eviction
• non-temporal access instructions

Determining the physical addresses aggressor/victim rows
• `/proc/self/pagemap`
• 2MB huge pages (relative)

But does it work on ARM?

(and we tried)
DMA

Direct Memory Access

Android’s DMA memory allocator provides everything we need:
• Uncached memory (no \texttt{clflush} required)
• Physically contiguous memory
DMA

Direct Memory Access

Android’s DMA memory allocator provides everything we need:
• Uncached memory (no **clflush** required)
• Physically contiguous memory

Physical memory:
DMA

Direct Memory Access

Android’s DMA memory allocator provides everything we need:
• Uncached memory (no clflush required)
• Physically contiguous memory

Physical memory:

DMA ALLOCATED CHUNK
DMA

Direct Memory Access

Android’s DMA memory allocator provides everything we need:
• Uncached memory (no `clflush` required)
• Physically contiguous memory

Physical memory:
DMA

Direct Memory Access

Android’s DMA memory allocator provides everything we need:

- Uncached memory (no `clflush` required)
- Physically contiguous memory

Physical memory:
Android’s DMA memory allocator provides everything we need:
• Uncached memory (no clflush required)
• Physically contiguous memory

Physical memory:
DMA

Direct Memory Access

Android’s DMA memory allocator provides everything we need:
• Uncached memory (no \texttt{clflush} required)
• Physically contiguous memory

Physical memory:
Android’s DMA memory allocator provides everything we need:
- Uncached memory (no `clflush` required)
- Physically contiguous memory

Physical memory:

![Bit Flip Diagram]
1. Memory Templating
Scan memory for useful bit flips

2. Land sensitive data
Store a crucial data structure on a vulnerable page

3. Reproduce the bit flip
Modify the data structure and get root access
Overview

1. Memory Templating
   Scan memory for useful bit flips

2. **Land sensitive data**
   Store a crucial data structure on a vulnerable page
Overview

1. Memory Templating
   Scan memory for useful bit flips

2. Land a Page Table
   Store a page table on a vulnerable page

But why?
Page Tables

Mapping virtual addresses to physical addresses
Page Tables

Mapping virtual addresses to physical addresses

Example lookup for input virtual address $0xb6a5717f$

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
</table>

Drammer: Deterministic Rowhammer Attacks on Mobile Platforms
Page Tables

Mapping virtual addresses to physical addresses

Example lookup for input virtual address \(0xb6a5717f\)

```
1 0 1 1 0 1 1 0 1 0 1 0 0 1 0 1 0 1 1 1 0 0 0 1 0 1 1 1 1 1 1
```

- Highest 12 bits: *level 1 table index* (\textit{Translation Table Base Register})
Mapping virtual addresses to physical addresses

Example lookup for input virtual address $0xb6a5717f$

```
1 0 1 1 0 1 1 0 1 0 1 0 0 1 0 1 0 1 1 1 0 0 0 1 0 1 1 1 1 1 1
```

- Highest 12 bits: *level 1 table index* (*Translation Table Base Register*)
- Middle 8 bits: *level 2 table index*
Mapping virtual addresses to physical addresses

Example lookup for input virtual address \(0xb6a5717f\)

- Highest 12 bits: *level 1 table index* (*Translation Table Base Register*)
- Middle 8 bits: *level 2 table index*
- Lowest 12 bits: *offset in page*
Mapping virtual addresses to physical addresses

Example lookup for input virtual address 0xb6a5717f

- Highest 12 bits: level 1 table index (Translation Table Base Register)
- Middle 8 bits: level 2 table index
- Lowest 12 bits: offset in page
Entry in the (2\textsuperscript{nd} level) Page Table

| 0 0 0 1 | 1 0 1 1 | 0 0 0 1 | 0 1 1 1 | 1 1 1 1 | x x x x | x x x x | x x x x |
Entry in the (2\textsuperscript{nd} level) Page Table

| 0 0 0 1 | 1 0 1 1 | 0 0 0 1 | 0 1 1 1 | 1 1 1 1 | \* \* \* \* | \* \* \* \* | \* \* \* \* |

- 12 bits of \textit{properties}
Entry in the (2\textsuperscript{nd} level) Page Table

- 12 bits of \textit{properties}
- 20 bits for the \textit{page base address}
Entry in the (2\textsuperscript{nd} level) Page Table

- 12 bits of \textit{properties}
- 20 bits for the \textit{page base address}

What if we flip a bit in the entry?
Entry in the (2\textsuperscript{nd} level) Page Table

- 12 bits of \textit{properties}
- 20 bits for the \textit{page base address}

0x1b17f000

\textit{mapped page}
Rowhammer Attacks on Page Table Entries

Entry in the (2nd level) Page Table

- 12 bits of properties
- 20 bits for the page base address

0x1b17f000
0x1b17e000
0x1b17f000

0x1b17f << 12
0x1b17e << 12
0x1b17f000

mapped page  mapped page

0 0 0 1 1 0 1 1 0 0 0 1 0 1 1 1 1 1 1 x x x x x x x x x x x x 
0 0 0 1 1 0 1 1 0 0 0 1 0 1 1 1 1 1 0 x x x x x x x x x x x x 
0 0 0 1 1 0 1 1 0 0 0 1 0 1 1 1 1 1 0 x x x x x x x x x x x x 
0 0 0 1 1 0 1 1 0 0 0 1 0 1 1 1 1 1 0 x x x x x x x x x x x x 
0 0 0 1 1 0 1 1 0 0 0 1 0 1 1 1 1 1 0 x x x x x x x x x x x x 
0 0 0 1 1 0 1 1 0 0 0 1 0 1 1 1 1 1 0 x x x x x x x x x x x x 
Entry in the (2\textsuperscript{nd} level) Page Table

- 12 bits of \textit{properties}
- 20 bits for the \textit{page base address}

A 1-to-0 flip moves the mapping ‘to the left’

- Flip offset 0: −1 page
- Flip offset 1: −2 pages
- Flip offset 2: −4 pages
- Flip offset \(n\): −\(2^n\) pages
1. Map a page 4 pages ‘away’ from its page table
Deterministic Attacks on Page Table Entries

1. Map a page 4 pages ‘away’ from its page table

```
0xb17b000 0xb17c000 0xb17d000 0xb17e000 0xb17f000
```

Page Table

```

```

Mapped Page
Deterministic Attacks on Page Table Entries

1. Map a page 4 pages ‘away’ from its page table

Virtual address 0xb6a57000 maps to Page Table Entry:

```
0 0 0 1 1 0 1 1 0 0 0 1 0 1 1 1 1 1 1 x x x x x x x x
```

which translates to physical page 0x1b17f000
Deterministic Attacks on Page Table Entries

1. Map a page 4 pages ‘away’ from its page table
2. Flip bit 2 in the page table entry

Virtual address 0xb6a57000 maps to Page Table Entry:

```
0 0 0 1 1 0 1 1 0 0 0 1 0 1 1 1 1 1 1 1 1
```

which translates to physical page 0x1b17f000
Deterministic Attacks on Page Table Entries

1. Map a page 4 pages ‘away’ from its page table
2. Flip bit 2 in the page table entry

Virtual address 0xb6a57000 maps to Page Table Entry:

```
0 0 0 1 1 0 1 1 0 0 0 1 0 1 1 1 1 0 1 x x x x x x x x x x x x x x
```

which translates to physical page 0x1b17b000
Deterministic Attacks on Page Table Entries

1. Map a page 4 pages ‘away’ from its page table
2. Flip bit 2 in the page table entry
3. Write page table entries

Virtual address 0xb6a57000 maps to Page Table Entry:

```
0 0 0 1 1 0 1 1 0 0 0 1 0 1 1 1 1 0 1 x x x x x x x x x x
```

which translates to physical page 0x1b17b000
Deterministic Attacks on Page Table Entries

1. Map a page 4 pages ‘away’ from its page table
2. Flip bit 2 in the page table entry
3. Write page table entries

Virtual address 0xb6a57000 maps to Page Table Entry:

| 0 0 0 1 | 1 0 1 1 | 0 0 0 1 | 0 1 1 1 | 1 1 0 1 | x x x x | x x x x | x x x x |

which translates to physical page 0x1b17b000
Deterministic Attacks on Page Table Entries

1. Map a page 4 pages ‘away’ from its page table
2. Flip bit 2 in the page table entry
3. Write page table entries
4. Read/write kernel memory

Virtual address 0xb6a57000 maps to Page Table Entry:

which translates to physical page 0x1b17b000
Deterministic Attacks on Page Table Entries

1. Map a page 4 pages ‘away’ from its page table
2. Flip bit 2 in the page table entry
3. Write page table entries
4. Read/write kernel memory
Overview

1. Memory Templating
   Scan memory for useful bit flips

2. Land a Page Table
   Store a page table on a vulnerable page

   But how?
Landing a Page Table

- No access to \texttt{pagemap} (virtual – physical address mapping)
- No fancy memory management features (deduplication)
Landing a Page Table

- No access to `pagemap` (virtual – physical address mapping)
- No fancy memory management features (deduplication)

Phys Feng Shui
Landing a Page Table

- No access to \texttt{pagemap} (virtual – physical address mapping)
- No fancy memory management features (deduplication)

Phys Feng Shui

Physical memory:
Landing a Page Table

- No access to `pagemap` (virtual – physical address mapping)
- No fancy memory management features (deduplication)

**Phys Feng Shui**

Physical memory:

Exhaust all memory
Drammer: Deterministic Rowhammer Attacks on Mobile Platforms

Landing a Page Table

- No access to `pagemap` (virtual – physical address mapping)
- No fancy memory management features (deduplication)

Phys Feng Shui

Physical memory:

Exhaust all memory
Drammer: Deterministic Rowhammer Attacks on Mobile Platforms

Landing a Page Table

- No access to *pagemap* (virtual – physical address mapping)
- No fancy memory management features (deduplication)

**Phys Feng Shui**

Physical memory:

Release the vulnerable page
Landing a Page Table

- No access to `pagemap` (virtual – physical address mapping)
- No fancy memory management features (deduplication)

Phys Feng Shui

Physical memory:

Release the vulnerable page
Landing a Page Table

- No access to pagemap (virtual – physical address mapping)
- No fancy memory management features (deduplication)

Phys Feng Shui

Physical memory:

Trigger a Page Table Allocation
Landing a Page Table

- No access to pagemap (virtual – physical address mapping)
- No fancy memory management features (deduplication)

Phys Feng Shui

Physical memory:

Trigger a Page Table Allocation
Drammer: Deterministic Rowhammer Attacks on Mobile Platforms

**Phys Feng Shui**

Exploit the predictable behavior of the **Buddy Allocator**

<table>
<thead>
<tr>
<th>Physical Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 * 4KB pages = 64 KB rows</td>
</tr>
</tbody>
</table>
Avoid fragmentation by keeping track of same-size memory chunks (*buddies*)

**Physical Memory**

16 * 4KB pages = 64 KB rows
Avoid fragmentation by keeping track of same-size memory chunks (buddies)
Avoid fragmentation by keeping track of same-size memory chunks (*buddies*).

\[ X_1 = \text{\_\_get\_free\_pages}(\text{flags}, \ 6); \quad \text{// get } 2^6 = 64\text{KB of memory} \]
Avoid fragmentation by keeping track of same-size memory chunks (buddies)

```c
X1 = get_free_pages(flags, 6); // get \(2^6 = 64\text{KB}\) of memory
```
Avoid fragmentation by keeping track of same-size memory chunks (*buddies*).

```c
X1 = __get_free_pages(flags, 6); // get $2^6 = 64$KB of memory
```
<table>
<thead>
<tr>
<th>Memory Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024KB</td>
</tr>
<tr>
<td>64KB</td>
</tr>
<tr>
<td>64KB</td>
</tr>
<tr>
<td>128KB</td>
</tr>
<tr>
<td>256KB</td>
</tr>
</tbody>
</table>

Avoid fragmentation by keeping track of same-size memory chunks (*buddies*)

```c
X1 = __get_free_pages(flags, 6); // get $2^6 = 64$KB of memory
```
Avoid fragmentation by keeping track of same-size memory chunks (buddies)

\[ X_1 = \text{__get_free_pages}(\text{flags}, 6); \quad // \quad \text{get } 2^6 = 64\text{KB of memory} \]
Avoid fragmentation by keeping track of same-size memory chunks (*buddies*)

\[x_2 = \text{__get_free_pages(flags, 3); } // \text{ get } 2^3 = 8\text{KB of memory}\]
Avoid fragmentation by keeping track of same-size memory chunks (*buddies*)

\[ X_2 = \texttt{__get_free_pages(flags, 3);} \quad \text{// get } 2^3 = 8\text{KB of memory} \]
Avoid fragmentation by keeping track of same-size memory chunks (*buddies*)

```c
X2 = __get_free_pages(flags, 3); // get $2^3 = 8$KB of memory
```
Avoid fragmentation by keeping track of same-size memory chunks (buddies)

\[ X^2 = \_\_\_\_g\_e\_t\_f\_r\_e\_p\_p\_a\_g\_e\_s(f\l_a\_g\_s, \ 3) ; \ // \ \text{get } 2^3 = 8\text{KB of memory} \]
Avoid fragmentation by keeping track of same-size memory chunks (buddies)

\[
x_2 = \text{\_get\_free\_pages}(\text{flags}, 3); \quad \text{// get } 2^3 = 8\text{KB of memory}
\]
Avoid fragmentation by keeping track of same-size memory chunks (*buddies*).

```c
X3 = __get_free_pages(flags, 5);  // get $2^3 = 32$KB of memory
```

## Buddy Allocator

<table>
<thead>
<tr>
<th>X2</th>
<th>8KB</th>
<th>16KB</th>
<th>32KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>128KB</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>256KB</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Avoid fragmentation by keeping track of same-size memory chunks (*buddies*).

\[ P_3 = \text{__get_free_pages}(\text{flags}, 5); \quad \text{// get } 2^3 = 32\text{KB of memory} \]
Avoid fragmentation by keeping track of same-size memory chunks (*buddies*).

```c
free_pages(X2, 3); // free X2
```
Avoid fragmentation by keeping track of same-size memory chunks (*buddies*)

```c
free_pages(X2, 3); // free X2
```
Avoid fragmentation by keeping track of same-size memory chunks (buddies)

```free_pages(X2, 3); // free X2```
Avoid fragmentation by keeping track of same-size memory chunks (*buddies*)

```c
free_pages(X2, 3); // free X2
```
Deterministic Rowhammer exploitation in 8 steps
Exhaust + Template \textit{Large} chunks

\texttt{L1, L2, \ldots, Ln = exhaust(L);}
Exhaust + Template \textit{Large} chunks

L1, L2, ..., Ln = \texttt{exhaust}(9); // get all $2^9 = 512$KB chunks
Exhaust + Template *Large* chunks

$L_1, L_2, \ldots, L_n = \text{exhaust}(L); \quad // \quad \text{get all } 2^9 = 512\text{KB chunks}$
Exhaust + Template Large chunks

`Hammer(L1, 2); // hammer row 2 of chunk L1`
Exhaust + Template *Large* chunks

Hammer(L1, 3); // hammer row 3 of chunk L1
Exhaust + Template *Large* chunks

\[
\text{Hammer}(L1, 4); // \text{hammer row 4 of chunk } L1
\]
Exhaust + Template \textit{Large} chunks

\texttt{Hammer(L1, 5); // hammer row 5 of chunk L1}
Exhaust + Template *Large* chunks

**Hammer**(L1, 6); // hammer row 6 of chunk L1
Exhaust + Template **Large** chunks

Hammer(L1, 7); // hammer row 7 of chunk L1
Exhaust + Template Large chunks

Hammer(L2, 2); // hammer row 2 of chunk L2
Exhaust + Template *Large* chunks

\textbf{Hammer(L2, 3);} // \texttt{hammer row 3 of chunk L2}
**Exhaust + Template Large chunks**

“exploitable flip found in page 5 of virtual row 3 of L2!”
Exhaust *Medium-sized* chunks

\[
_M1, _M2, \ldots, _Mn = \text{exhaust}(6); \quad // \text{get all } 2^6 = 64\text{KB chunks}
\]
Exhaust Medium-sized chunks

\_M1, \_M2, \ldots, \_Mn = exhaust(6); // get all 2^6 = 64KB chunks
Exhaust *Medium-sized* chunks

\[_M1, _M2, \ldots, _Mn = \text{exhaust}(6); // get all } 2^6 = 64\text{KB chunks}\]
Exhaust *Medium-sized* chunks

\_M1, \_M2, \ldots, \_Mn = exhaust(6); // get all $2^6 = 64\text{KB}$ chunks
Exhaust *Medium-sized* chunks

\_M1, \_M2, \ldots, \_Mn = \text{exhaust}(6); \quad /\!\!/ \text{get all } 2^6 = 64\text{KB chunks}
Exhaust *Medium-sized* chunks

_{M1}, _{M2}, \ldots, _{Mn} = \text{exhaust}(6); // get all $2^6 = 64\text{KB}$ chunks
Exhaust *Medium-sized* chunks

\[ \_M1, \_M2, \ldots, \_Mn = \text{exhaust}(6); \] // get all \(2^6 = 64\text{KB}\) chunks
Exhaust *Medium-sized* chunks

_\_M1, _\_M2, ..., _\_Mn = exhaust(6);  // get all 2^6 = 64KB chunks
Exhaust *Medium-sized* chunks

\[ _M1, _M2, \ldots, _Mn = \text{exhaust}(6); \quad // \quad \text{get all } 2^6 = 64\text{KB chunks} \]
Exhaust *Medium-sized* chunks

\[M_1, M_2, \ldots, M_n = \text{exhaust}(6); \]  // get all \(2^6 = 64\text{KB}\) chunks
Exhaust *Medium-sized* chunks

\[ \_M1, \_M2, \ldots, \_Mn = \text{exhaust}(6); \] // get all \(2^6 = 64\text{KB}\) chunks
Release *Large* chunk with vulnerable page

Release (L2); // L chunk with vulnerable page
Exhaust *Medium-sized* chunks (again)

\[ M_1, M_2, ..., M_n = \text{exhaust}(6); \] \text{ // get all } 2^6 = 64KB \text{ chunks}
Exhaust *Medium-sized* chunks (again)

\[ M_1, M_2, \ldots, M_n = \text{exhaust}(6); \quad \text{// get all } 2^6 = 64\text{KB chunks} \]
Exhaust *Medium-sized* chunks (again)

\[ M_1, M_2, \ldots, M_n = \text{exhaust}(6); // \text{get all } 2^6 = 64\text{KB} \text{ chunks} \]
Exhaust *Medium-sized* chunks (again)

\[
M_1, M_2, \ldots, M_n = \text{exhaust}(6); \quad // \quad \text{get all } 2^6 = 64\text{KB chunks}
\]
Exhaust *Medium-sized* chunks (again)

\[
M_1, M_2, \ldots, M_n = \text{exhaust}(6); \quad // \quad \text{get all } 2^6 = 64\text{KB chunks}
\]
Exhaust *Medium-sized* chunks (again)

\[
M_1, M_2, \ldots, M_n = \text{exhaust}(6); \quad // \quad \text{get all } 2^6 = 64\text{KB chunks}
\]
Exhaust *Medium-sized* chunks (again)

\[ M_1, M_2, \ldots, M_n = \text{exhaust}(6); \quad \text{// get all } 2^6 = 64\text{KB chunks} \]
Exhaust *Medium-sized* chunks (again)

```c
M1, M2, ..., Mn = exhaust(6); // get all $2^6 = 64$KB chunks
```
Exhaust *Medium-sized* chunks (again)

\[ M_1, M_2, \ldots, M_n = \text{exhaust}(6); \quad // \quad \text{get all } 2^6 = 64\text{KB chunks} \]
Exhaust *Medium-sized* chunks (again)

\[
M_1, M_2, \ldots, M_n = \text{exhaust}(6); \quad // \quad \text{get all } 2^6 = 64\text{KB chunks}
\]
Exhaust *Medium-sized* chunks (again)

```plaintext
M1, M2, ..., Mn = exhaust(6); // get all 2^6 = 64KB chunks
```
Exhaust *Medium-sized* chunks (again)

\[
M_1, M_2, \ldots, M_n = \text{exhaust}(6); \quad // \text{get all } 2^6 = 64\text{KB chunks}
\]
Exhaust *Medium-sized* chunks (again)

\[ M_1, M_2, \ldots, M_n = \text{exhaust}(6); \]  // get all \(2^6 = 64\text{KB}\) chunks
Exhaust *Medium-sized* chunks (again)

M1, M2, ..., Mn = exhaust(6); // get all $2^6 = 64$KB chunks
Exhaust *Medium-sized* chunks (again)

\[ M_1, M_2, \ldots, M_n = \text{exhaust}(6); \quad // \text{get all } 2^6 = 64\text{KB chunks} \]
Exhaust *Medium-sized* chunks (again)

```c
M1, M2, ..., Mn = exhaust(6); // get all 2^6 = 64KB chunks
```
Release vulnerable *Medium-sized* chunk + Release all Large chunks
Release vulnerable *Medium-sized* chunk + Release all Large chunks

Release(M3);  // releases the vulnerable row
Release vulnerable *Medium-sized* chunk + Release all Large chunks

ReleaseAll(L); // to avoid going out-of-memory later
Land a *small* chunk in the vulnerable 64 KB row

\texttt{Land(S); // allocate 4KB pages until the 64KB is used}
Land a *small* chunk in the vulnerable 64 KB row

\texttt{Land(S); // allocate 4KB pages until the 64KB is used}
Land a *small* chunk in the vulnerable 64 KB row

```plaintext
Land(S); // allocate 4KB pages until the 64KB is used
```
Land a *small* chunk in the vulnerable 64 KB row

\texttt{Land(S); // allocate 4KB pages until the 64KB is used}
Land a *small* chunk in the vulnerable 64 KB row

```plaintext
Land(S); // allocate 4KB pages until the 64KB is used
```
Land a *small* chunk in the vulnerable 64 KB row

\[ \text{Land}(S); \quad // \text{allocate 4KB pages until the 64KB is used} \]
Land a *small* chunk in the vulnerable 64 KB row

\[ \text{Land(S); // allocate 4KB pages until the 64KB is used} \]
Land a *small* chunk in the vulnerable 64 KB row

```plaintext
Land(S); // allocate 4KB pages until the 64KB is used
```
Land a *small* chunk in the vulnerable 64 KB row

\[ \text{Land}(S) \; ; \; // \; \text{allocate 4KB pages until the 64KB is used} \]
Land a small chunk in the vulnerable 64 KB row

\texttt{Land(S);} // allocate 4KB pages until the 64KB is used
Land a *small* chunk in the vulnerable 64 KB row

\[\text{Land}(S)\; \text{// allocate 4KB pages until the 64KB is used}\]
Land a *small* chunk in the vulnerable 64 KB row

\texttt{Land(S); // allocate 4KB pages until the 64KB is used}
Land a *small* chunk in the vulnerable 64 KB row

```
Land(S); // allocate 4KB pages until the 64KB is used
```
Land a *small* chunk in the vulnerable 64 KB row

\texttt{Land(S); // allocate 4KB pages until the 64KB is used}
Land a *small* chunk in the vulnerable 64 KB row

\texttt{Land(S); // allocate 4KB pages until the 64KB is used}
Land a *small* chunk in the vulnerable 64 KB row

\[ \text{Land(S); } \quad \text{// allocate 4KB pages until the 64KB is used} \]
Land a *small* chunk in the vulnerable 64 KB row

Land(S);  // allocate 4KB pages until the 64KB is used
Land a *small* chunk in the vulnerable 64 KB row

\( \text{Land}(S) \); // allocate 4KB pages until the 64KB is used
Land a *small* chunk in the vulnerable 64 KB row

`Land(S); // allocate 4KB pages until the 64KB is used`
Land a small chunk in the vulnerable 64 KB row

\texttt{Land(S); // allocate 4KB pages until the 64KB is used}
Land a *small* chunk in the vulnerable 64 KB row

\[ \text{Land}(S) ; \quad \text{allocate 4KB pages until the 64KB is used} \]
Pad small chunks until the vulnerable page

\texttt{Pad(P); // insert padding until vulnerable page}
Pad small chunks until the vulnerable page

\texttt{Pad(P); // insert padding until vulnerable page}
Pad *small* chunks until the vulnerable page

Pad(P); // insert padding until vulnerable page
Pad small chunks until the vulnerable page

\[ \text{Pad}(P); \quad \text{// insert padding until vulnerable page} \]
Pad small chunks until the vulnerable page

\texttt{Pad(P); // insert padding until vulnerable page}
Force a Page Table allocation + map the vulnerable PTE

\[
PT = \text{mmap}(\text{MAP\_FIXED}); \quad // \quad \text{Force a Page Table allocation}
\]
Force a Page Table allocation + map the vulnerable PTE

\[ \text{PT} = \text{mmap(MAP\_FIXED);} \] // Force a Page Table allocation
Force a Page Table allocation + map the vulnerable PTE

\[ \text{PT} = \text{mmap(MAP\_FIXED);} \]
Drammer: Deterministic Rowhammer Attacks on Mobile Platforms

Phys Feng Shui step 8/8

Force a Page Table allocation + map the vulnerable PTE

\[ PT = \text{mmap}(\text{MAP\_FIXED}); \] // Force a Page Table allocation
Force a Page Table allocation + map the vulnerable PTE
Drammer: Deterministic Rowhammer Attacks on Mobile Platforms

Phys Feng Shui step 8/8

Force a Page Table allocation + **map the vulnerable PTE**
**Force a Page Table allocation + map the vulnerable PTE**

Drammer: Deterministic Rowhammer Attacks on Mobile Platforms

Phys Feng Shui step 8/8

<table>
<thead>
<tr>
<th>M1</th>
<th>M2</th>
<th>M3</th>
<th>M4</th>
<th>M5</th>
<th>M6</th>
<th>M7</th>
<th>M8</th>
</tr>
</thead>
<tbody>
<tr>
<td>S9</td>
<td>P1</td>
<td>P2</td>
<td>P3</td>
<td>PT</td>
<td>4KB</td>
<td>8KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

PTE with bit flip

8KB (first page)
Force a Page Table allocation + map the vulnerable PTE

16 * 4KB pages = 64KB rows
Force a Page Table allocation + map the vulnerable PTE

```c
mmap(M4[5], MAP_FIXED); // map vulnerable PTE 64KB ‘away’
```

16 * 4KB pages = 64KB rows
Drammer: Deterministic Rowhammer Attacks on Mobile Platforms

Overview

1. Memory Templating
   Scan memory for useful bit flips

2. Land a Page Table
   Store a page table on a vulnerable page

3. Reproduce the bit flip
   Modify the data structure and get root access
Drammer

Perform double-sided rowhammer to flip a bit in the PTE

16 * 4KB pages = 64KB rows
Perform double-sided rowhammer to flip a bit in the PTE

16 * 4KB pages = 64KB rows
Perform double-sided rowhammer to flip a bit in the PTE

<table>
<thead>
<tr>
<th>M2</th>
<th>M4 [5]</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2</td>
<td>P3</td>
</tr>
<tr>
<td>PT</td>
<td>4KB</td>
</tr>
<tr>
<td>8KB</td>
<td>4KB</td>
</tr>
</tbody>
</table>

16 * 4KB pages = 64KB rows
### Drammer

**Write access to a Page Table**

16 * 4KB pages = 64KB rows

<table>
<thead>
<tr>
<th></th>
<th>M2</th>
<th>P2</th>
<th>P3</th>
<th>PT</th>
<th>4KB</th>
<th>8KB (first page)</th>
</tr>
</thead>
<tbody>
<tr>
<td>M4[3]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(3rd page)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M4[4]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(4th page)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M4[5]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(5th page)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M4[6]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(6th page)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M4[7]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(7th page)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1. Fill PT with Page Table Entries to kernel memory
Drammer

Write access to a Page Table

16 * 4KB pages = 64KB rows

<table>
<thead>
<tr>
<th>16 * 4KB pages = 64KB rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2</td>
</tr>
<tr>
<td>P2</td>
</tr>
<tr>
<td>P3</td>
</tr>
<tr>
<td>PT</td>
</tr>
<tr>
<td>4KB</td>
</tr>
<tr>
<td>8KB (first page)</td>
</tr>
</tbody>
</table>

1. Fill PT with Page Table Entries to kernel memory
2. Search kernel memory for our `struct cred`
Drammer
Write access to a Page Table

16 * 4KB pages = 64KB rows

<table>
<thead>
<tr>
<th>M2</th>
<th>P2</th>
<th>P3</th>
<th>PT</th>
<th>4KB</th>
<th>8KB (first page)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>PT</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. Fill PT with Page Table Entries to kernel memory
2. Search kernel memory for our `struct cred`
3. Overwrite our `uid` and `gid` to get root privileges
## Evaluation

<table>
<thead>
<tr>
<th>Device</th>
<th>#flips</th>
<th>1st exploitable flip after</th>
</tr>
</thead>
<tbody>
<tr>
<td>LG Nexus 5¹</td>
<td>1058</td>
<td>116s</td>
</tr>
<tr>
<td>LG Nexus 5⁴</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>LG Nexus 5⁵</td>
<td>747,013</td>
<td>1s</td>
</tr>
<tr>
<td>LG Nexus 4</td>
<td>1,328</td>
<td>7s</td>
</tr>
<tr>
<td>OnePlus One</td>
<td>3,981</td>
<td>942s</td>
</tr>
<tr>
<td>Motorola Moto G (2013)</td>
<td>429</td>
<td>441s</td>
</tr>
<tr>
<td>LG G4 (ARMv8 – 64-bit)</td>
<td>117,496</td>
<td>5s</td>
</tr>
</tbody>
</table>

Bit flips on 18 out of 27 tested devices
Evaluation

<table>
<thead>
<tr>
<th>Device</th>
<th>#flips</th>
<th>1st exploitable flip after</th>
</tr>
</thead>
<tbody>
<tr>
<td>LG Nexus 5¹</td>
<td>1058</td>
<td>116s</td>
</tr>
<tr>
<td>LG Nexus 5⁴</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>LG Nexus 5⁵</td>
<td>747,013</td>
<td>1s</td>
</tr>
<tr>
<td>LG Nexus 4</td>
<td>1,328</td>
<td>7s</td>
</tr>
<tr>
<td>OnePlus One</td>
<td>3,981</td>
<td>942s</td>
</tr>
<tr>
<td>Motorola Moto G (2013)</td>
<td>429</td>
<td>441s</td>
</tr>
<tr>
<td>LG G4 (ARMv8 – 64-bit)</td>
<td>117,496</td>
<td>5s</td>
</tr>
</tbody>
</table>
### Evaluation

<table>
<thead>
<tr>
<th>Device</th>
<th>#flips</th>
<th>$1^{st}$ exploitable flip after</th>
</tr>
</thead>
<tbody>
<tr>
<td>LG Nexus 5(^1)</td>
<td>1058</td>
<td>116s</td>
</tr>
<tr>
<td>LG Nexus 5(^4)</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>LG Nexus 5(^5)</td>
<td>747,013</td>
<td>1s</td>
</tr>
<tr>
<td>LG Nexus 4</td>
<td>1,328</td>
<td>7s</td>
</tr>
<tr>
<td>OnePlus One</td>
<td>3,981</td>
<td>942s</td>
</tr>
<tr>
<td>Motorola Moto G (2013)</td>
<td>429</td>
<td>441s</td>
</tr>
<tr>
<td>LG G4 (ARMv8 – 64-bit)</td>
<td>117,496</td>
<td>5s</td>
</tr>
<tr>
<td>Device</td>
<td>#flips</td>
<td>1&lt;sup&gt;st&lt;/sup&gt; exploitable flip after</td>
</tr>
<tr>
<td>------------------------------</td>
<td>---------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>LG Nexus 5&lt;sup&gt;1&lt;/sup&gt;</td>
<td>1058</td>
<td>116s</td>
</tr>
<tr>
<td>LG Nexus 5&lt;sup&gt;4&lt;/sup&gt;</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>LG Nexus 5&lt;sup&gt;5&lt;/sup&gt;</td>
<td>747,013</td>
<td>1s</td>
</tr>
<tr>
<td>LG Nexus 4</td>
<td>1,328</td>
<td>7s</td>
</tr>
<tr>
<td>OnePlus One</td>
<td>3,981</td>
<td>942s</td>
</tr>
<tr>
<td>Motorola Moto G (2013)</td>
<td>429</td>
<td>441s</td>
</tr>
<tr>
<td>LG G4 (ARMv8 – 64-bit)</td>
<td>117,496</td>
<td>5s</td>
</tr>
</tbody>
</table>
### Evaluation

<table>
<thead>
<tr>
<th>Device</th>
<th>#flips</th>
<th>1st exploitable flip after</th>
</tr>
</thead>
<tbody>
<tr>
<td>LG Nexus 5&lt;sup&gt;1&lt;/sup&gt;</td>
<td>1058</td>
<td>116s</td>
</tr>
<tr>
<td>LG Nexus 5&lt;sup&gt;4&lt;/sup&gt;</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>LG Nexus 5&lt;sup&gt;5&lt;/sup&gt;</td>
<td>747,013</td>
<td>1s</td>
</tr>
<tr>
<td>LG Nexus 4</td>
<td>1,328</td>
<td>7s</td>
</tr>
<tr>
<td>OnePlus One</td>
<td>3,981</td>
<td>942s</td>
</tr>
<tr>
<td>Motorola Moto G (2013)</td>
<td>429</td>
<td>441s</td>
</tr>
<tr>
<td>LG G4 (ARMv8 – 64-bit)</td>
<td>117,496</td>
<td>5s</td>
</tr>
</tbody>
</table>

After the 1<sup>st</sup> exploitable flip, exploitation takes at most 22 seconds.
Evaluation

<table>
<thead>
<tr>
<th>Device</th>
<th>#flips</th>
<th>1&lt;sup&gt;st&lt;/sup&gt; exploitable flip after</th>
</tr>
</thead>
<tbody>
<tr>
<td>LG Nexus 5&lt;sup&gt;1&lt;/sup&gt;</td>
<td>1058</td>
<td>116s</td>
</tr>
<tr>
<td>LG Nexus 5&lt;sup&gt;4&lt;/sup&gt;</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>LG Nexus 5&lt;sup&gt;5&lt;/sup&gt;</td>
<td>747,013</td>
<td>1s</td>
</tr>
<tr>
<td>LG Nexus 4</td>
<td>1,328</td>
<td>7s</td>
</tr>
<tr>
<td>OnePlus One</td>
<td>3,981</td>
<td>942s</td>
</tr>
<tr>
<td>Motorola Moto G (2013)</td>
<td>429</td>
<td>441s</td>
</tr>
<tr>
<td>LG G4 (ARMv8 – 64-bit)</td>
<td>117,496</td>
<td>5s</td>
</tr>
</tbody>
</table>

After the 1<sup>st</sup> exploitable flip, exploitation takes at most 22 seconds.

Drammer test app reported bit flips on: Google Pixel, OnePlus 3, Galaxy Note 7, HTC One M8, ...
Disclosure

Contacted Google with a list of suggested mitigations on July 25
Contacted Google with a list of suggested mitigations on July 25
(91 days before #CCS16)
Disclosure

Contacted Google with a list of suggested mitigations on July 25

(91 days before #CCS16)

“Can you publish at another conference, later this year?”
Disclosure

Contacted Google with a list of suggested mitigations on July 25

(91 days before #CCS16)

“Can you publish at another conference, later this year?”

“What if we support you financially?”
Disclosure

Contacted Google with a list of suggested mitigations on July 25

(91 days before #CCS16)

“Ok, could you then perhaps obfuscate some parts of the paper?”
Disclosure

Contacted Google with a list of suggested mitigations on July 25

(91 days before #CCS16)

“Ok, could you then perhaps obfuscate some parts of the paper?”

Rewarded $4000 for a critical issue
Disclosure

Contacted Google with a list of suggested mitigations on July 25
(91 days before #CCS16)

“Ok, could you then perhaps obfuscate some parts of the paper?”

Rewarded $4000 for a critical issue
(because “it doesn’t work on the devices in our Reward Program”)

Drammer: Deterministic Rowhammer Attacks on Mobile Platforms
Disclosure

Contacted Google with a list of suggested mitigations on July 25
(91 days before #CCS16)

“Ok, could you then perhaps obfuscate some parts of the paper?”

Rewarded $4000 for a critical issue
(because “it doesn’t work on the devices in our Reward Program”)

But now it does
Disclosure

Contacted Google with a list of suggested mitigations on July 25
(91 days before #CCS16)

“Ok, could you then perhaps obfuscate some parts of the paper?”

Rewarded $4000 for a critical issue

Partial hardening in November’s updates

“We will continue to work on a longer term solution”
Conclusion

• Deterministic Rowhammer exploitation
• **No special memory management features required** (e.g., deduplication)
• ARM memory controllers are fast enough to do Rowhammer
• LPDDR* found vulnerable
• No easy software fix
Conclusion

• Deterministic Rowhammer exploitation
• No special memory management features required (e.g., deduplication)
• ARM memory controllers are fast enough to do Rowhammer
• LPDDR* found vulnerable
• No easy software fix
• Using DMA bypasses state-of-the-art defenses (e.g., ANVIL)
Conclusion

• Deterministic Rowhammer exploitation
• No special memory management features required (e.g., deduplication)
• ARM memory controllers are fast enough to do Rowhammer
• LPDDR* found vulnerable
• No easy software fix
• Using DMA bypasses state-of-the-art defenses (e.g., ANVIL)

• More details
  • Demos, statistics and test app:
    https://vusec.net/projects/drammer
  • Open source:
    https://github.com/vusec/drammer