DRAMA: How your DRAM becomes a security problem

Michael Schwarz and Anders Fogh
November 4, 2016
This talk is about how DRAM leaks information across security boundaries

• Not about software bugs
• It is about hardware design becomes an attack vector
• Focus on Intel x86-64 - but problem is DRAM - thus applies to other architectures as well
Take aways

- DRAM design is security relevant
- DRAM leaks information
Take aways

• DRAM design is security relevant
• DRAM leaks information

Exploit this to:

• Covertly extract information cross VM, cross CPU
• Spy on other software
• Enable efficient and targeted row hammer attacks
Introduction
What you just saw

- 0 software bugs
- Covert communication in and out of VM
- Covert communication in and out of JS sandbox
- This isn’t magic..
• Michael Schwarz
• PhD Student, Graz University of Technology
• Likes to break stuff
• Twitter: @misc0110
• Email: michael.schwarz@iaik.tugraz.at
And the team

The research team

• Peter Pessl
• Daniel Gruss
• Clémentine Maurice
• Stefan Mangard

from Graz University of Technology
Whoami

• Anders Fogh
• Principal Security Researcher, GDATA Advanced Analytics
• Playing with malware since 1992
• Twitter: @anders_fogh
• Email: anders.fogh@gdata-adan.de
From code to capacitor
MOV RAX, [0x414141414141]

TO

[Image of a capacitor]
Page tables
Virtual and physical addressing

• 0x41414141 is a virtual address of the current process
• The CPU need a physical address to talk to DRAM
• Thus translation is needed
Why translation

Why address translation: Run multiple processes securely on a single CPU

• Let applications run in their own virtual address space
• Create exchangeable map from “virtual memory” to “physical memory”
• Privileges are checked on memory accesses
• Managed by the operating system kernel and hypervisor
Address translation on x86-64

- CR3
- PML4
  - PML4E 0
  - PML4E 1
  - ...
  - #PML4I
  - PML4E 511
- PDPT
  - PDPTE 0
  - PDPTE 1
  - ...
  - #PDPTI
  - PDPTE 511
- Page Directory
  - PDE 0
  - PDE 1
  - ...
  - PDE #PDI
  - PDE 511
- Page Table
  - PTE 0
  - PTE 1
  - ...
  - PTE #PTI
  - PTE 511
- 4 KiB Page
  - Byte 0
  - Byte 1
  - ...
  - Offset
  - Byte 4095

- PML4I (9 b)
- PDPTI (9 b)
- PDI (9 b)
- PTI (9 b)
- Offset (12 b)
Virtual and physical addressing

Summary:

• The most significant bits of the virtual address determines the page
• A page is almost always 4 kB large
• The least significant bits (almost always 12 bits) is an offset into the page
• Only the page is translated and security checked
Data caches
Road block: Data Caches

Memory (DRAM) is slow compared to the CPU

• buffer frequently used memory for the CPU
• every memory reference goes through the cache
• transparent to OS and programs

= Problem: We want to speak to DRAM, not a cache
We must remove our address from the cache to talk to DRAM

- Native code: CLFLUSH instruction
The memory controller
How does physical addresses map to memory

Memory controller in the processor has a mapping function

Based on physical addresses
How is DRAM organized?
How is DRAM organized?

channel 0

channel 1
How is DRAM organized?

Channel 0

Back of DIMM: rank 1

Front of DIMM: rank 0

Channel 1
How is DRAM organized?

channel 0

channel 1

back of DIMM: rank 1

front of DIMM: rank 0

chip
DRAM organization

chip

bank 0

| row 0 |
| row 1 |
| row 2 |
| ... |
| row 32767 |

row buffer
DRAM organization

chip

bank 0

row 0
row 1
row 2
... 
row 32767

row buffer

64k Cells
1 Capacitor,
1 transistor each
First hint of trouble

- A row is 64k Cells: 8kB
- Security was checked for 4kB blocks
First hint of trouble

• A row is 64k Cells: 8 kB
• Security was checked for 4 kB blocks
  = security domains may share rows
Reading from DRAM
The Row buffer

• DRAM internally is only capable of reading entire rows
The Row buffer

- DRAM internally is only capable of reading entire rows
- Capacitors in cells discharge when you “read the bits”
- Buffer the bits when reading them from the cells
- Write the bits back to the cells when you’re done
The Row buffer

• DRAM internally is only capable of reading entire rows
• Capacitors in cells discharge when you “read the bits”
• Buffer the bits when reading them from the cells
• Write the bits back to the cells when you’re done

= Row buffer
How reading from DRAM works

CPU reads row 1, row buffer empty!
How reading from DRAM works
How reading from DRAM works

The figure illustrates how reading from DRAM works. The DRAM bank contains multiple rows of memory cells. When the CPU requests data, it provides an address to the row buffer, which selects the corresponding row in the DRAM bank. The row is then read into the row buffer and the data is returned to the CPU.
How reading from DRAM works

DRAM bank

CPU reads row 1, row buffer now full!
How reading from DRAM works
We can measure a difference

**Figure 1:** Row hits
We can measure a difference

Figure 2: Row conflicts
We can measure a difference

Difference between row hits ($\approx 225$ cycles) and row conflicts ($\approx 247$ cycles) on an Intel Core i7 Ivy Bridge machine.
• Security is typically checked for 4 kB pages
• Security is typically checked for 4kB pages
• The data caches can be circumvented to use DRAM
• Security is typically checked for 4 kB pages
• The data caches can be circumvented to use DRAM
• 4 kB pages of different security domains can share banks
• Security is typically checked for 4 kB pages
• The data caches can be circumvented to use DRAM
• 4 kB pages of different security domains can share banks
• 4 kB pages of different security domains can share rows
Summary

• Security is typically checked for 4 kB pages
• The data caches can be circumvented to use DRAM
• 4 kB pages of different security domains can share banks
• 4 kB pages of different security domains can share rows
• Through timing we can establish row hits and misses across security domains
• Security is typically checked for 4 kB pages
• The data caches can be circumvented to use DRAM
• 4 kB pages of different security domains can share banks
• 4 kB pages of different security domains can share rows
• Through timing we can establish row hits and misses across security domains

= DRAM leaks information
First attack: Reversing the CPU
Memory controller in the processor has a mapping function

- Based on physical addresses
Memory controller in the processor has a mapping function

- Based on physical addresses
- Problem: this function is undocumented
Reversing the mapping function

Reverse engineer the mapping function

• You can reverse engineer the mapping of your processor using row hits and misses
Reversing the mapping function - Approach

Different bank

Same bank
Reversing the mapping function - Approach

Select random base address in one bank

Different bank  Same bank
Reversing the mapping function - Approach

Measure access time when repeatedly accessing base and random address

Different bank

Same bank
Reversing the mapping function - Approach

Measure access time when repeatedly accessing base and random address

Different bank  Same bank
Reversing the mapping function - Approach

Measure access time when repeatedly accessing base and random address

Different bank

Same bank

15
13
16
Reversing the mapping function - Approach

Measure access time when repeatedly accessing base and random address

Different bank

Same bank
Measure access time when repeatedly accessing base and random address
Reversing the mapping function - Approach

Measure access time when repeatedly accessing base and random address

Different bank

Same bank
Reversing the mapping function - Approach

Measure access time when repeatedly accessing base and random address.

Different bank

Same bank

45
43
46
15
13
16
4
Reversing the mapping function - Approach

Measure access time when repeatedly accessing base and random address.

Different bank

Same bank

45
43
46
15
13
16

1
4
Measure access time when repeatedly accessing base and random address

Different bank

Same bank

Reversing the mapping function - Approach
Reversing the mapping function - Approach

Select random base address in one bank

Different bank

Same bank
Reversing the mapping function - Approach

Measure access time when repeatedly accessing base and random address

Different bank

Same bank

16
Measure access time when repeatedly accessing base and random address.
Reversing the mapping function - Approach

Measure access time when repeatedly accessing base and random address

Different bank  
Same bank
Reversing the mapping function - Approach

Measure access time when repeatedly accessing base and random address

Different bank

Same bank

15
13
16

46
Reversing the mapping function - Approach

Measure access time when repeatedly accessing base and random address

Different bank

Same bank

15
13
16

43
46

42
43
44
45
46
47

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
Reversing the mapping function - Approach

Measure access time when repeatedly accessing base and random address

Different bank

Same bank

15
13
16

45
43
46

42
43
44
45
46
47
Reversing the mapping function - Approach

Measure access time when repeatedly accessing base and random address

Different bank

Same bank
Reversing the mapping function - Approach

Measure access time when repeatedly accessing base and random address

Different bank

Same bank
Reversing the mapping function - Approach

Measure access time when repeatedly accessing base and random address
Repeat the process for all banks
Reversing the mapping function - Approach

- Repeat the process for all banks
- For each bank, we have a set of addresses that map to this bank
Reversing the mapping function - Approach

- Repeat the process for all banks
- For each bank, we have a set of addresses that map to this bank
- We can see it as a linear equation system
Reversing the mapping function - Approach

- Repeat the process for all banks
- For each bank, we have a set of addresses that map to this bank
- We can see it as a linear equation system
- Solving it gives us the bits used for the mapping functions
Reversing the mapping function - Approach

• Repeat the process for all banks
• For each bank, we have a set of addresses that map to this bank
• We can see it as a linear equation system
• Solving it gives us the bits used for the mapping functions
• The alternative: generate every possible XOR function and check if it yields the same result for all addresses in the set
Reversing the mapping function - Approach

• Repeat the process for all banks
• For each bank, we have a set of addresses that map to this bank
• We can see it as a linear equation system
• Solving it gives us the bits used for the mapping functions
• The alternative: generate every possible XOR function and check if it yields the same result for all addresses in the set
• This is still very fast (in the order of seconds)
We developed a toolkit that reverse engineers the mapping fully automatically.
• We developed a toolkit that reverse engineers the mapping fully automatically
• Takes between seconds and minutes
• We developed a toolkit that reverse engineers the mapping fully automatically
• Takes between seconds and minutes
• You can download it here: https://github.com/IAIK/drama
• We know which address maps to which part of the DRAM
What next?

• We know which address maps to which part of the DRAM
• We can do that fully automatic on any new system
What next?

• We know which address maps to which part of the DRAM
• We can do that fully automatic on any new system
• Once we have the function, we can exploit that knowledge
What next?

Stay Tuned
For something Awesome
Spying through the DRAM
WCHAR WideCharFromScanCode(unsigned short scancode)
{
    if (IsUpperCase(scancode))
    {
        return HandleUpperCase(scancode);
    }
    else
    {
        return HandleLowerCase(scancode);
    }
}
• We want to spy on the behaviour of a victim
• We want to spy on the behaviour of a victim
• The victim will not know that we spy on it
• We want to spy on the behaviour of a victim
• The victim will not know that we spy on it
• We can use row hits to get useful information
Spying

• We want to spy on the behaviour of a victim
• The victim will not know that we spy on it
• We can use row hits to get useful information
• Advantage over cache attacks: it works across CPUs
Attacks

Attack Primitive: Row hit

DRAM bank

Spy activates row 0, get copied to row buffer

row buffer
Attacks

Attack Primitive: Row hit

![Diagram of DRAM bank with rows highlighted and activation and copy arrows]
Attacks

Attack Primitive: Row hit
Attacks

Attack Primitive: Row hit

Spy activates shared row

activate

copy
Attacks

Attack Primitive: Row hit

Row conflict, high timing
Attacks

Attack Primitive: Row hit

...but what if the victim accessed the shared row...
Attacks

Attack Primitive: Row hit

![Diagram of DRAM bank with row buffer and activation process]

- **DRAM bank**
  - Rows are represented by 0s and 1s.
  - Activation and copying are indicated by respective arrows.

- **Row buffer**
  - Contains the data that is copied from the activated row.

- **Activate**
  -标识用于激活特定行的机制。

- **Copy**
  -标识用于从激活行复制数据的机制。

This diagram illustrates the row hit attack, showing how data is activated and copied from one row to another within a DRAM bank.
Attacks

Attack Primitive: Row hit

...before the spy activates it
Attacks

Attack Primitive: Row hit

Row hit, faster
Two related questions

• What is the chance we can share a row with important victim data?
Two related questions

• What is the chance we can share a row with important victim data?
• What kind of spatial accuracy will we get?
Row hits

• The smallest unit of physical memory is one page.
• Pages are usually 4 kB.
• DRAM rows are usually 8 kB.
• We need the victim's address and our address in the same row.
• The smallest unit of physical memory is one page
• The smallest unit of physical memory is one page
• Pages are usually 4 kB
Row hits

- The smallest unit of physical memory is one page
- Pages are usually 4 kB
- DRAM rows are usually 8 kB
Row hits

- The smallest unit of physical memory is one page
- Pages are usually 4 kB
- DRAM rows are usually 8 kB
- We need the victim’s address and our address in the same row
Row hits

- The smallest unit of physical memory is one page
- Pages are usually 4 kB
- DRAM rows are usually 8 kB
- We need the victim’s address and our address in the same row
Accuracy

- If you say that **two pages** share one row you are not wrong...
• If you say that two pages share one row you are not wrong...

• ...but not right either
• If you say that *two pages* share one row you are not wrong...
• ...but not right either
• Why?
• Not the whole physical page must be in one row
Accuracy

• Not the whole physical page must be in one row
• Depending on the mapping function, a page can be distributed over multiple rows
• Not the whole physical page must be in one row
• Depending on the mapping function, a page can be distributed over multiple rows
• This is the case if address bits 0 to 11 are used for the mapping
• Not the whole physical page must be in one row
• Depending on the mapping function, a page can be distributed over multiple rows
• This is the case if address bits 0 to 11 are used for the mapping
• For example: Skylake uses low bits for channel (bits 8 and 9) and bankgroup (bit 7)
• Not the whole physical page must be in one row
• Depending on the mapping function, a page can be distributed over multiple rows
• This is the case if address bits 0 to 11 are used for the mapping
• For example: Skylake uses low bits for channel (bits 8 and 9) and bankgroup (bit 7)
• One physical page is distributed over 4 rows
Accuracy

8 kB row $x$ in BG0 (1) and channel (1)

8 kB row $x$ in BG0 (0) and channel (1)

8 kB row $x$ in BG0 (1) and channel (0)

8 kB row $x$ in BG0 (0) and channel (0)
Accuracy

8 kB row $x$ in BG0 (1) and channel (1)

8 kB row $x$ in BG0 (0) and channel (1)

8 kB row $x$ in BG0 (1) and channel (0)

8 kB row $x$ in BG0 (0) and channel (0)
Results

Row sharing

Sandy Bridge /w 1 DIMM

2 pages per row
Row sharing

Ivy Bridge /w 2 DIMM

DRAM bank

4 pages per row
Row sharing

Sky Lake /w 2 DIMM

DRAM bank

8 pages per row
Row sharing

DRAM bank

Sky Lake /w 2 DIMM

8 pages per row

row buffer
• We can deduct behavior from memory access much like cache side channel attacks
• We can deduct behavior from memory access much like cache side channel attacks
• Works cross VM, cross CPU or sandboxed!
Summary

• We can deduct behavior from memory access much like cache side channel attacks
• Works cross VM, cross CPU or sandboxed!
• On the latest generation of personal computers
• We can deduce behavior from memory access much like cache side channel attacks
• Works cross VM, cross CPU or sandboxed!
• On the latest generation of personal computers
  • We are likely to be in the same row as secret victim information
• We can deduct behavior from memory access much like cache side channel attacks
• Works cross VM, cross CPU or sandboxed!
• On the latest generation of personal computers
  • We are likely to be in the same row as secret victim information
  • We have a spatial accuracy of 1024 bytes
• We can deduce behavior from memory access much like cache side channel attacks
• Works cross VM, cross CPU or sandboxed!
• On the latest generation of personal computers
  • We are likely to be in the same row as secret victim information
  • We have a spatial accuracy of 1024 bytes
  • It gets even better on multi-CPU servers
• We can deduct behavior from memory access much like cache side channel attacks
• Works cross VM, cross CPU or sandboxed!
• On the latest generation of personal computers
  • We are likely to be in the same row as secret victim information
  • We have a spatial accuracy of 1024 bytes
  • It gets even better on multi-CPU servers
• For example, we can spy on keyboard inputs to Firefox
But Wait!!!---That’s NOT All
DRAM Covert Channel
Attacks

Attack Primitive: Row miss

Sender and receiver decide on one bank

DRAM bank

row buffer
Attack Primitive: Row miss

Receiver measures access time to its address
Attack Primitive: Row miss
Attacks

Attack Primitive: Row miss

Repeated access always has low access times
Attacks

Attack Primitive: Row miss
Attacks

Attack Primitive: Row miss

Sender accesses its address

activate

copy
Attacks

Attack Primitive: Row miss

Sender accesses its address
Attacks

Attack Primitive: Row miss

On next access of receiver, there is a row miss.
Attacks

Attack Primitive: Row miss
How our demo really works

What is a covert communication?
How our demo really works

What is a covert communication?

- Two programs would like to communicate
How our demo really works

What is a covert communication?

- Two programs would like to communicate but are not allowed to do so
How our demo really works

What is a covert communication?

- Two programs would like to communicate but are not allowed to do so
- All “normal” channels are blocked or monitored
How our demo really works

What is a covert communication?

• Two programs would like to communicate but are not allowed to do so
• All “normal” channels are blocked or monitored

EXPLOIT

ALL THE SIDE CHANNELS
We are “trapped” inside a VM without network hardware

• There is no communication between guest and host
Scenario

We are “trapped” inside a VM without network hardware

• There is no communication between guest and host
• We want to get data out of the VM
We are “trapped” inside a VM without network hardware

- There is no communication between guest and host
- We want to get data out of the VM
- We cannot run binaries on the host system
Scenario

We are “trapped” inside a VM without network hardware

• There is no communication between guest and host
• We want to get data out of the VM
• We cannot run binaries on the host system
• There are no known software bugs in either host, guest or virtualization software
The solution

A covert channel implemented in JavaScript

- DRAM as side channel (main memory is “shared” between host and guest)
- Sender inside the VM
- JavaScript running in the browser on the host
- We only have to trick the victim to visit our page
A covert channel implemented in JavaScript
The solution

A covert channel implemented in JavaScript

• DRAM as side channel (main memory is “shared” between host and guest)
A covert channel implemented in JavaScript

- DRAM as side channel (main memory is “shared” between host and guest)
- Sender inside the VM
A covert channel implemented in JavaScript

- DRAM as side channel (main memory is “shared” between host and guest)
- Sender inside the VM
- JavaScript running in the browser on the host
The solution

A covert channel implemented in JavaScript

- DRAM as side channel (main memory is “shared” between host and guest)
- Sender inside the VM
- JavaScript running in the browser on the host
- We only have to trick the victim to visit our page
• Use the row miss attack primitive
• Use the row miss attack primitive
• Sender and receiver agree on a bank (can be hardcoded)
The gory details - bits

- Use the row miss attack primitive
- Sender and receiver agree on a bank (can be hardcoded)
- Both sender inside VM and JavaScript in host select a different row inside this bank
The gory details - bits

- Use the row miss attack primitive
- Sender and receiver agree on a bank (can be hardcoded)
- Both sender inside VM and JavaScript in host select a different row inside this bank
- JavaScript measures access time for this row
• Use the row miss attack primitive
• Sender and receiver agree on a bank (can be hardcoded)
• Both sender inside VM and JavaScript in host select a different row inside this bank
• JavaScript measures access time for this row
• Sender can transmit 0 by doing nothing and 1 by causing row conflict
The gory details - bits

• Use the row miss attack primitive
• Sender and receiver agree on a bank (can be hardcoded)
• Both sender inside VM and JavaScript in host select a different row inside this bank
• JavaScript measures access time for this row
• Sender can transmit 0 by doing nothing and 1 by causing row conflict
• If measured timing was “fast” sender transmitted 0.
The gory details - bits

Figure 3: Multiple measurements per bit to have a reliable detection.
The gory details - bits

Figure 3: Multiple measurements per bit to have a reliable detection.
• Communication is based on packets
The gory details - Packets

- Communication is based on packets
- Packet starts with a 2-bit preamble
• Communication is based on packets
• Packet starts with a 2-bit preamble
• Data integrity is checked by an error-detection code (EDC)
Communication is based on packets
Packet starts with a 2-bit preamble
Data integrity is checked by an error-detection code (EDC)
Sequence bit indicates whether it is a retransmission or a new packet
• Transmission of approximately 11 bits/s
Results

- Transmission of approximately 11 bits/s
- Can be improved using
Results

- Transmission of approximately 11 bits/s
- Can be improved using
  - Fewer retransmits
Results

- Transmission of approximately 11 bits/s
- Can be improved using
  - Fewer retransmits
  - Error correction
Results

- Transmission of approximately 11 bits/s
- Can be improved using
  - Fewer retransmits
  - Error correction
  - Multithreading → multiple banks in parallel

What is possible in native code?

596 kbit/s cross CPU and cross VM
Results

- Transmission of approximately 11 bits/s
- Can be improved using
  - Fewer retransmits
  - Error correction
  - Multithreading → multiple banks in parallel
  - What is possible in native code? 596 kbit/s cross CPU and cross VM
Rowhammer
• Capacitors leak $\rightarrow$ refresh necessary
• cells leak faster upon proximate accesses
• With enough proximate access bits flips
• Capacitors leak $\rightarrow$ refresh necessary
• cells leak faster upon proximate accesses
• With enough proximate access bits flips
Rowhammer

- Capacitors leak $\rightarrow$ refresh necessary
- Cells leak faster upon proximate accesses
- With enough proximate access bits flips
Rowhammer

- Capacitors leak → refresh necessary
- Cells leak faster upon proximate accesses
- With enough proximate access bits flips
Rowhammer

- Capacitors leak $\rightarrow$ refresh necessary
- Cells leak faster upon proximate accesses
- With enough proximate access bits flips
Rowhammer

- Capacitors leak → refresh necessary
- Cells leak faster upon proximate accesses
- With enough proximate access bits flips
The problem: Finding the victim row and the neighboring rows.
Solution 1: Spraying - We can fill memory with security relevant information and hammer randomly

- Seaborn 2015
- Spraying PTE and NaCl sanity checking code
- Problem: Not everything can be sprayed.
Solution 2: Deduplication

- Razavi et al. 2016
- We can have the operating system / hypervisor copy relevant information to a known location
- Problem: Deduplication is turned off in ”serious” cloud and default off in most operating systems.
Solution 3: Locate data - DRAMA: We know the mapping function

- Bhattacharya and Mukhopadhyay 2016
- Cool: We can now target row hammer
- Problem: Physical addresses.
Solution 3: Locate data - DRAMA: We know the mapping function

- Bhattacharya and Mukhopadhyay 2016
- Cool: We can now target row hammer
- Problem: Physical addresses.
- /proc/PID/pagemap
- cite prefetch
- Other leaks: ex. large pages and cache set congruency.
Knowing the mapping function and physical address is what enabled bit flips in DDR4
Solution 4: Locate data - DRAMA: Row hits and misses

- If we can invoke victim:
- We can use row miss primitive to locate the bank
- We can use row hits primitive to locate rows
Finding rows

Solution 4: Locate data - DRAMA: Row hits and misses

- If we can invoke victim:
- We can use row miss primitive to locate the bank
- We can use row hits primitive to locate rows
- This is not perfect,
- but we can drastically improve accuracy
Conclusion
Black Hat Sound Bytes.

- DRAM design is security relevant
- We can covertly exfiltrate information
- We can spy on other software
- We enable targeted row hammer attacks
References


Gruss, Daniel et al. (2016). “Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript”. In: DIMVA.


DRAMA: How your DRAM becomes a security problem

Michael Schwarz and Anders Fogh
November 4, 2016