Rowhammer: From the Basics to Sophisticated New Variants

Moritz Lipp, Michael Schwarz
February 20, 2018
Graz University of Technology
Who are we

- **Moritz Lipp**
  - PhD Student @ Graz University of Technology
  - @mlqxyz
  - moritz.lipp@iaik.tugraz.at

- **Michael Schwarz**
  - PhD Student @ Graz University of Technology
  - @misc0110
  - michael.schwarz@iaik.tugraz.at
Security and privacy rely on secrets (unknown to attackers)
• Security and privacy rely on secrets (unknown to attackers)

• Secrets can leak through side channels
Software-based Side-Channel Attacks

- Security and privacy rely on secrets (unknown to attackers)
- Secrets can leak through side channels
- Software-based attacks $\rightarrow$ no physical access
Software-based side-channel attacks leak secrets via...

- Algorithm runtimes
- Cache states
- DRAM states
- ...
• Rowhammer has a special rule among microarchitectural attacks
• Rowhammer has a special rule among microarchitectural attacks
• Does not leak secrets...
• Rowhammer has a special rule among microarchitectural attacks
• Does not leak secrets...
• ...but it can change memory contents
• Rowhammer has a special rule among microarchitectural attacks
• Does not leak secrets...
• ...but it can change memory contents
• A software-based fault attack on the DRAM
Background
printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);

Cache miss
printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);
DRAM access, slow

printf("%d", i);
printf("%d", i);

Cache miss

Cache hit

DRAM access, slow

Request
Response

Cache miss

Cache hit
printf("%d", i);
Cache miss
DRAM access, slow

printf("%d", i);
Cache hit
No DRAM access, much faster

printf("%d", i);
Request
Response
Memory Access Latency

The chart illustrates the distribution of memory access latencies. The x-axis represents latency in cycles, ranging from 0 to 500+. The y-axis shows the number of accesses, with values ranging from 1 to 1,000,000.

There are two categories shown:
- **Cached**
- **Not Cached**

The data indicates that a significant portion of accesses have latencies within the lower cycles, with a peak around 50 cycles. The number of accesses decreases as latency increases, with fewer accesses in the higher latency ranges.
DRAM organization

channel 0

channel 1

back of DIMM: rank 1

front of DIMM:

rank 0

Moritz Lipp, Michael Schwarz — Graz University of Technology
DRAM organization

channel 0

channel 1
DRAM organization

channel 0

back of DIMM: rank 1

channel 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
• DRAM internally is only capable of reading entire rows
• DRAM internally is only capable of reading entire rows
• Capacitors in cells discharge when reading them
• DRAM internally is only capable of reading entire rows
• Capacitors in cells discharge when reading them
• Bits are buffered when reading them from the cells
The row buffer

- DRAM internally is only capable of reading entire rows
- Capacitors in cells discharge when reading them
- Bits are buffered when reading them from the cells
- Then, bits are written back to the cells again
• DRAM internally is only capable of reading entire rows
• Capacitors in cells discharge when reading them
• Bits are buffered when reading them from the cells
• Then, bits are written back to the cells again
• → Row buffer
Read from DRAM

DRAM bank

CPU reads row 1:
→ activate and copy to row buffer
Reading from DRAM

DRAM bank

activate

copy

row buffer

1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1

. . .

1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1

...
Reading from DRAM

DRAM bank

111111111111111
111111111111111
111111111111111
111111111111111
... 
111111111111111
row buffer

return

Moritz Lipp, Michael Schwarz — Graz University of Technology
Reading from DRAM

CPU reads row 2:
→ activate and copy to row buffer
→ row conflict → slow
Reading from DRAM

DRAM bank

1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
... 
1 1 1 1 1 1 1 1 1 1 1 1 1 1

row buffer

return
Reading from DRAM

CPU reads row 2:
→ already in row buffer
→ row hit → fast
Reading from DRAM

CPU reads row 2:
→ already in row buffer
→ row hit → fast
Row hits (≈ 225 cycles) and row conflicts (≈ 247 cycles)
Rowhammer
- Cells leak → repetitive refresh necessary
- Maximum interval between refreshes to guarantee data integrity
- Cells leak faster upon proximate accesses → Rowhammer
Cells leak → repetitive refresh necessary

Maximum interval between refreshes to guarantee data integrity

Cells leak faster upon proximate accesses → Rowhammer
● Cells leak → repetitive refresh necessary
● Maximum interval between refreshes to guarantee data integrity
● Cells leak faster upon proximate accesses → Rowhammer
- Cells leak → repetitive refresh necessary
- Maximum interval between refreshes to guarantee data integrity
- Cells leak faster upon proximate accesses → Rowhammer
• Cells leak $\rightarrow$ repetitive *refresh* necessary

• Maximum interval between refreshes to guarantee *data integrity*

• Cells leak faster upon proximate accesses $\rightarrow$ *Rowhammer*
Rowhammer

- Cells leak $\rightarrow$ repetitive refresh necessary
- Maximum interval between refreshes to guarantee data integrity
- Cells leak faster upon proximate accesses $\rightarrow$ Rowhammer
How widespread is the issue?

DDR3:
- Kim et al.: 110/129 modules from 3 vendors, all but 3 since mid-2011
- Seaborn and Dullien: 15/29 laptops

DDR4 believed to be safe:
- we showed bit flips (Pessl et al. 2016)

Prevalence, by Kim et al. 2014
1. Flush cache via `clflush` instruction → original paper (Kim et al. 2014)
3. Non-temporal accesses (Qiao et al. 2016)
4. Uncached memory (Veen et al. 2016)
#1 Hammering with clflush

cache set 1

cache set 2

DRAM bank
Hammering with clflush

cache set 1

cache set 2

clflush

clflush

DRAM bank
#1 Hammering with clflush

- DRAM bank
- cache set 1
- cache set 2
- clflush
- clflush

Moritz Lipp, Michael Schwarz — Graz University of Technology
Hammering with clflush

Cache set 1

Cache set 2

DRAM bank
#1 Hammering with clflush

DRAM bank

reload

cache set 1

cache set 2
#1 Hammering with clflush

- cache set 1
- cache set 2
- DRAM bank
Hammering with **clflush**

- **cache set 1**
- **cache set 2**
- **DRAM bank**

---

Moritz Lipp, Michael Schwarz — Graz University of Technology
Hammering with clflush

DRAM bank

reload

cache set 1

cache set 2
Hammering with clflush
#1 Hammering with `clflush`

[Diagram showing cache sets and DRAM bank with reload arrows between cache sets and DRAM bank.]
Hammering with clflush

cache set 1

cache set 2

DRAM bank

clflush

clflush
Hammering with `clflush`

- Cache set 1
- Cache set 2
- DRAM bank

Reload

Moritz Lipp, Michael Schwarz — Graz University of Technology
Hammering with clflush

Cache set 1

Cache set 2

DRAM bank

wait for it...
Hammering with clflush

DRAM bank

cache set 1

cache set 2

reload

bit flip!
#2 Hammering with cache eviction

DRAM bank

cache set 1

cache set 2

Moritz Lipp, Michael Schwarz — Graz University of Technology
#2 Hammering with cache eviction

DRAM bank

cache set 1

cache set 2

load

load
#2 Hammering with cache eviction

- Cache set 1
- Cache set 2
- DRAM bank

Moritz Lipp, Michael Schwarz — Graz University of Technology
Hammering with cache eviction

- Cache set 1
- Cache set 2
- DRAM bank

Moritz Lipp, Michael Schwarz — Graz University of Technology
#2 Hammering with cache eviction

DRAM bank

cache set 1

cache set 2

load

load
#2 Hammering with cache eviction

DRAM bank

[Diagram showing cache sets and loads]

Load

Cache set 1

Cache set 2

Moritz Lipp, Michael Schwarz — Graz University of Technology
#2 Hammering with cache eviction

DRAM bank

cache set 1

load

load

cache set 2
Hammering with cache eviction

Cache set 1

Cache set 2

DRAM bank
#2 Hammering with cache eviction

**cache set 1**

**cache set 2**

**DRAM bank**
Hammering with cache eviction

- Cache set 1
- Cache set 2
- DRAM bank

Reload
Hammering with cache eviction

cache set 1

cache set 2

DRAM bank

repeat!
#2 Hammering with cache eviction

- cache set 1
- cache set 2
- DRAM bank

reload

wait for it...
Hammering with cache eviction

Cache set 1

Cache set 2

DRAM bank

bit flip!
There are three different hammering techniques

#1: Hammer one row next to victim row and other random rows

#2: Hammer two rows neighboring victim row

#3: Hammer only one row next to victim row
Single-sided hammering

DRAM bank

activate
#1 - Single-sided hammering

DRAM bank

activate
#1 - Single-sided hammering

![Diagram of a DRAM bank with binary values and an arrow labeled "activate" pointing to a specific row.]
#1 - Single-sided hammering

![Diagram showing DRAM bank with binary values and activation process.]

Moritz Lipp, Michael Schwarz — Graz University of Technology
#1 - Single-sided hammering

![Diagram of a DRAM bank with binary values and an activation arrow]

Moritz Lipp, Michael Schwarz — Graz University of Technology
#1 - Single-sided hammering

[Diagram of a DRAM bank with bit flips indicated]
#2 - Double-sided hammering

![Diagram of DRAM bank with activation marks]

Moritz Lipp, Michael Schwarz — Graz University of Technology
#2 - Double-sided hammering

![Diagram of a DRAM bank with specific rows highlighted to illustrate the concept of double-sided hammering.](image-url)
#2 - Double-sided hammering

![DRAM bank diagram]

 activates

Moritz Lipp, Michael Schwarz — Graz University of Technology
#2 - Double-sided hammering

activate

DRAM bank

1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
#2 - Double-sided hammering

![DRAM bank diagram](image)

**activate**
#2 - Double-sided hammering

DRAM bank

activate

bit flips

Moritz Lipp, Michael Schwarz — Graz University of Technology
#3 - One-location hammering

DRAM bank

activate
#3 - One-location hammering

![DRAM bank diagram](image-url)
#3 - One-location hammering

DRAM bank

activate

1111111111111111
1111111111111111
1111111111111111
1111111111111111
1111111111111111
1111111111111111
1111111111111111
1111111111111111
#3 - One-location hammering

DRAM bank

1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
#3 - One-location hammering

DRAM bank

activate
#3 - One-location hammering

![Diagram of DRAM bank with bit flips highlighted]
• We proposed one-location hammering
• We proposed one-location hammering
• The only technique which accesses only one row
• We proposed one-location hammering
• The only technique which accesses only one row
• Circumvents all countermeasures which expect accesses to multiple rows
• We proposed one-location hammering
• The only technique which accesses only one row
• Circumvents all countermeasures which expect accesses to multiple rows
• Why does it work?
• **Open-page policy**: Keep row opened and buffered
• **Open-page policy**: Keep row opened and buffered
  • Beneficial for memory access latency, power consumption, bank utilization for a low number of memory accesses
• **Open-page policy**: Keep row opened and buffered
  - Beneficial for memory access latency, power consumption, bank utilization for a low number of memory accesses

• **Close-page policy**: Immediately close row, ready to open a new row
• **Open-page policy**: Keep row opened and buffered
  • Beneficial for memory access latency, power consumption, bank utilization for a low number of memory accesses

• **Close-page policy**: Immediately close row, ready to open a new row
  • Achieve better performance when a high number of memory accesses is performed
• **Open-page policy**: Keep row opened and buffered
  - Beneficial for memory access latency, power consumption, bank utilization for a low number of memory accesses

• **Close-page policy**: Immediately close row, ready to open a new row
  - Achieve better performance when a high number of memory accesses is performed
  - Perform better on multi-core systems David et al. 2011
• Policies that preemptively close rows, would allow one-location hammering
• Policies that preemptively close rows, would allow one-location hammering
• We observed close-page policies on desktop computers
Policies that preemptively close rows, would allow one-location hammering.

We observed close-page policies on desktop computers.

Mobile devices (e.g., laptops) seem to use mostly open-page policies.
How well does it work?

- Distribution of bit flips over 4kb-aligned memory regions
- Test each technique for 8 hours
- Scanned for bit flips after every hammering attempt
  - Hammering a random location of more than 100 000 randomly-chosen 4kB pages
How well does it work?

Double-sided
77.0% bit offsets
51.7% 0→1 bit flips

Single-sided
78.5% bit offsets
54.1% 0→1 bit flips

One-location
36.5% bit offsets
51.6% 0→1 bit flips
Exploits
How to exploit random bit flips?

- They are not random $\rightarrow$ highly reproducible flip pattern!
  1. Choose a data structure that you can place at arbitrary memory locations
  2. Scan for "good" flips
  3. Place data structure there
  4. Trigger bit flip again
Each 4 KB page table consists of 512 such entries.
Each 4 KB page table consists of 512 such entries.
### Page Table Entries

<table>
<thead>
<tr>
<th>P</th>
<th>RW</th>
<th>US</th>
<th>WT</th>
<th>UC</th>
<th>R</th>
<th>D</th>
<th>S</th>
<th>G</th>
<th>Ignored</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Physical Page Number**

<table>
<thead>
<tr>
<th>Ignored</th>
<th>X</th>
</tr>
</thead>
</table>
Each 4 KB page table consists of 512 such entries
Getting root by targeting the kernel
Page Table Manipulation

```
<table>
<thead>
<tr>
<th>Address Range</th>
<th>PTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0 - 0xFFF</td>
<td>PTE 0</td>
</tr>
<tr>
<td>0x1000 - 0x1FFF</td>
<td>PTE 1</td>
</tr>
<tr>
<td>0x2000 - 0x2FFF</td>
<td>PTE 2</td>
</tr>
<tr>
<td>0x3000 - 0x3FFF</td>
<td>PTE 3</td>
</tr>
<tr>
<td>0x4000 - 0x4FFF</td>
<td>PTE 4</td>
</tr>
<tr>
<td>0x5000 - 0x5FFF</td>
<td>PTE 5</td>
</tr>
<tr>
<td>0x6000 - 0x6FFF</td>
<td>PTE 6</td>
</tr>
<tr>
<td>0x7000 - 0x7FFF</td>
<td>PTE 7</td>
</tr>
</tbody>
</table>
```

- `User Page`: Pages marked with a filled square are part of user memory.
- `Kernel Page`: Pages marked with a outlined square are part of kernel memory.
- `Page Table`: The page table contains entries for each page frame.
Page Table Manipulation

- PTE 7
- PTE 6
- PTE 5
- PTE 4
- PTE 3
- PTE 2
- PTE 1
- PTE 0

Address Ranges:
- 0x0 - 0xFFF
- 0x1000 - 0x1FFF
- 0x2000 - 0x2FFF
- 0x3000 - 0x3FFF
- 0x4000 - 0x4FFF
- 0x5000 - 0x5FFF
- 0x6000 - 0x6FFF
- 0x7000 - 0x7FFF

User Pages
- 0x0 - 0xFFF
- 0x1000 - 0x1FFF
- 0x2000 - 0x2FFF
- 0x3000 - 0x3FFF
- 0x4000 - 0x4FFF
- 0x5000 - 0x5FFF
- 0x6000 - 0x6FFF
- 0x7000 - 0x7FFF

Kernel Page
- 0x0 - 0xFFF

Page Table
Page Table Manipulation

0x7000 – 0x7FFF
0x6000 – 0x6FFF
0x5000 – 0x5FFF
0x4000 – 0x4FFF
0x3000 – 0x3FFF
0x2000 – 0x2FFF
0x1000 – 0x1FFF
0x0 – 0xFFF

PTE 0
PTE 1
PTE 2
PTE 3
PTE 4
PTE 5
PTE 6
PTE 7

User Page
User Page
User Page
User Page
User Page
User Page
User Page
Kernel Page
Page Table
User Page
User Page
Page Table Manipulation

- PTE 0: 0x0 - 0xFFF
- PTE 1: 0x1000 - 0x1FFF
- PTE 2: 0x2000 - 0x2FFF
- PTE 3: 0x3000 - 0x3FFF
- PTE 4: 0x4000 - 0x4FFF
- PTE 5: 0x5000 - 0x5FFF
- PTE 6: 0x6000 - 0x6FFF
- PTE 7: 0x7000 - 0x7FFF

- User Page
- User Page
- User Page
- User Page
- User Page
- User Page
- User Page
- Kernel Page
- Page Table
- User Page
Hammering memory locations in different rows

- Row 0
- Row 23
Hammering memory locations in different rows
Hammering memory locations in different rows
Hammering memory locations in different rows
Hammering memory locations in different rows
Hammering memory locations in different rows
Release bit flip page
Release bit flip page
Fill all remaining memory with page tables

Place page table on bit flip page
Fill all remaining memory with page tables

Place page table on bit flip page
Page Table Manipulation

Page Table

User Page

Page Table

Page Table

Page Table

Page Table

Kernel Page

Page Table

Page Table

Page Table

Page Table
Page Table Manipulation

Page Table
- PTE 7
- PTE 6
- PTE 5
- PTE 4
- PTE 3
- PTE 2
- PTE 1
- PTE 0

User Page
- Page Table
- Page Table
- Page Table
- Page Table
- Kernel Page
- Page Table
- Page Table
- Page Table

0x7000 – 0x7FFF
0x6000 – 0x6FFF
0x5000 – 0x5FFF
0x4000 – 0x4FFF
0x3000 – 0x3FFF
0x2000 – 0x2FFF
0x1000 – 0x1FFF
0x0 – 0xFFF
Page Table Manipulation

- Page Table
- Page Table
- User Page
- Page Table
- Page Table
- Kernel Page
- Page Table
- Page Table
- Page Table

Moritz Lipp, Michael Schwarz — Graz University of Technology
Strategy: Flipping Page Table PPN bits

1. Scan for flips
2. Exhaust or massage memory to place a page table at target location
3. Gain access to your own page table → kernel privileges
Idea from Seaborn et al. 2015
• Idea from Seaborn et al. 2015

• Same idea applied in several other works:
  • Rowhammer.js (Gruss, Maurice, et al. 2016)
  • One bit flips, one cloud flops (Xiao et al. 2016)
  • Drammer (Veen et al. 2016)
What if we cannot target kernel pages?
Many applications perform actions as root
• Many applications perform actions as root
• They can be used by unprivileged users as well
 Opcode Flipping

- Many applications perform actions as root
- They can be used by unprivileged users as well
- Implicitly: e.g., ping or mount
• Many applications perform actions as root
• They can be used by unprivileged users as well
• Implicitly: e.g., ping or mount
• Explicitly: sudo
● Target *sudo*, as it is the easiest to exploit
• Target `sudo`, as it is the easiest to exploit
• Use Rowhammer to circumvent password check
- Target `sudo`, as it is the easiest to exploit
- Use Rowhammer to circumvent password check
- Changing program logic with bit flips
• Target `sudo`, as it is the easiest to exploit
• Use Rowhammer to circumvent password check
• Changing program logic with bit flips
• What happens if we induce bit flips in instructions?
 Opcode Flipping - Conditional Jump

JE

0 1 1 1 0 1 0 0

↑

HLT

1 1 1 1 0 1 0 0
 Opcode Flipping - Conditional Jump

JE
0 1 1 1 0 1 0 0

→
XORB
0 0 1 1 0 1 0 0
Opcode Flipping - Conditional Jump

JE

PUSHQ
Opcode Flipping - Conditional Jump

JE

0 1 1 1 0 1 0 0

<prefix>

0 1 1 0 0 1 0 0

Moritz Lipp, Michael Schwarz — Graz University of Technology
Opcode Flipping - Conditional Jump

JE

\[
\begin{array}{cccccc}
0 & 1 & 1 & 1 & 0 & 1 \ 0 \ 0
\end{array}
\]

JL

\[
\begin{array}{cccccc}
0 & 1 & 1 & 1 & 1 & 1 \ 1 \ 0 \ 0
\end{array}
\]
Opcode Flipping - Conditional Jump

JE

0 1 1 1 0 1 0 0

JO

0 1 1 1 1 0 0 0

Moritz Lipp, Michael Schwarz — Graz University of Technology
Opcode Flipping - Conditional Jump

JE

JBE

0 1 1 1 0 1 0 0

0 1 1 1 0 1 1 0

Moritz Lipp, Michael Schwarz — Graz University of Technology
 Opcode Flipping - Conditional Jump

JE

0 1 1 1 0 1 0 0

JNE

0 1 1 1 0 1 0 1

Moritz Lipp, Michael Schwarz — Graz University of Technology
• Conditional jumps are not the only targets
• Conditional jumps are not the only targets
• Other targets include

Manual analysis of sudo revealed 29 possible bitflips
They all somehow skipped the password check
• Conditional jumps are not the only targets
• Other targets include
  • Comparisons
  • Addresses of memory loads/stores
  • Address calculations
  • ...
• Conditional jumps are not the only targets
• Other targets include
  • Comparisons
  • Addresses of memory loads/stores
  • Address calculations
  • ...
• Manual analysis of sudo revealed 29 possible bitflips
• Conditional jumps are not the only targets
• Other targets include
  • Comparisons
  • Addresses of memory loads/stores
  • Address calculations
  • ...
• Manual analysis of `sudo` revealed 29 possible bitflips
• They all somehow skipped the password check
• Remaining problem: maneuver the target binary page to a vulnerable physical page
Placing the binary

- Remaining problem: maneuver the target binary page to a vulnerable physical page
- Not as easy as with page tables
• Remaining problem: maneuver the target binary page to a vulnerable physical page
• Not as easy as with page tables
• Many page tables can simply be sprayed over memory
• Remaining problem: maneuver the target binary page to a vulnerable physical page
• Not as easy as with page tables
• Many page tables can simply be sprayed over memory
• There is only one copy of the binary in memory
• If a binary is loaded the first time, it is loaded to the memory
• If a binary is loaded the first time, it is loaded to the memory
• It stays in memory (in the page cache) even after execution
• If a binary is loaded the first time, it is loaded to the memory
• It stays in memory (in the page cache) even after execution
• Only evicted if page cache is full
• If a binary is loaded the first time, it is loaded to the memory
• It stays in memory (in the page cache) even after execution
• Only evicted if page cache is full
• Page cache is huge - usually all unused memory
We propose memory waylaying to move a victim page to a target page.
• We propose memory waylaying to move a victim page to a target page
• By filling the page cache with executable pages, we evict the victim binary
We propose memory waylaying to move a victim page to a target page

By filling the page cache with executable pages, we evict the victim binary

Using the mincore function, we can check if the victim binary is evicted
We propose memory waylaying to move a victim page to a target page. By filling the page cache with executable pages, we evict the victim binary. Using the `mincore` function, we can check if the victim binary is evicted. Loading the victim binary results in a new physical page.
• We propose memory waylaying to move a victim page to a target page
• By filling the page cache with executable pages, we evict the victim binary
• Using the \texttt{mincore} function, we can check if the victim binary is evicted
• Loading the victim binary results in a new physical page
• Continue until it is at the target page
(1) Start
(2) Evict Page Cache
(3) Access Binary
(4) Evict + Access
(5) Evict + Access
(6) Stop if target reached
How well does it work?

- New pages cover most of the physical memory
Great advantage over memory massaging: only negligible memory footprint
Other Techniques

- Modify binary pages executed in root privileges (Xiao et al. 2016)
- Modify credential structs (Veen et al. 2016)
- Read keys (Xiao et al. 2016)
- Corrupt RSA signatures (Bhattacharya et al. 2016)
- Modify certificates
- Configurations
- etc.
Rowhammer + SGX = Cheap Denial of Service
- SGX is an x86 instruction-set extension
- Integrity and Confidentiality of code and data in untrusted environments
- Run with user privileges and restricted, e.g., no system calls
- Execute programs in enclaves using protected areas of memory
SGX Encrypted Memory

Physical Memory

0 GB

16 GB

Moritz Lipp, Michael Schwarz — Graz University of Technology
• SGX explicitly protects against DRAM-based attacks
  • Cold-boot attacks
  • Memory bus snooping
  • Memory tampering attacks
• SGX EPC (Enclave Page Cache)
  • Region in DRAM
  • Cryptographically ensuring confidentiality, integrity and freshness of data
• Memory Encryption Engine (MEE)
  • Transparently encrypts memory
• What happens if a bit flips in the EPC?
• Integrity check will fail!
  → Lock’s up the memory controller
  → Not a single further memory access!
  → System halts immediately
What happens if a bit flips in the EPC?

- Integrity check will fail!
  - Lock’s up the memory controller
  - Not a single further memory access!
  - System halts immediately

Sounds unsafe?
What happens if a bit flips in the EPC?
- Integrity check will fail!
  - Lock’s up the memory controller
  - Not a single further memory access!
  - System halts immediately

Sounds unsafe? **It is unsafe!**
More and more services are deployed using SGX
• More and more services are deployed using SGX
• Cloud providers offer SGX support
  • Microsoft’s Azure Confidential Computing
• More and more services are deployed using SGX
• Cloud providers offer SGX support
  • Microsoft’s Azure Confidential Computing
• If a malicious enclave induces a bit flip, ...
• More and more services are deployed using SGX
• Cloud providers offer SGX support
  • Microsoft’s Azure Confidential Computing
• If a malicious enclave induces a bit flip, . . .
• . . . the entire machine halts
More and more services are deployed using SGX.

Cloud providers offer SGX support:

- Microsoft’s Azure Confidential Computing

If a malicious enclave induces a bit flip, . . .

. . . the entire machine halts

. . . including co-located tenants
• More and more services are deployed using SGX
• Cloud providers offer SGX support
  • Microsoft’s Azure Confidential Computing
• If a malicious enclave induces a bit flip, . . .
• . . . the entire machine halts
• . . . including co-located tenants
• Denial-of-Service Attacks in the Cloud
SGX + One-location Hammering + Opcode Flipping = Undetectable Exploit
SGX protects software from malicious environments.
• SGX protects software from malicious environments
• Thwarts static and dynamic (= performance counters) analysis
(Ab)using SGX Protection

- SGX protects software from malicious environments
- Thwarts static and dynamic (= performance counters) analysis
- Hammering from SGX defeats countermeasures relying on this:
  - MASCAT
  - ANVIL
  - HexPADS
  - Herath and Fogh
  - Gruss et al.
  - Zhang et al.
  - Chiappetta et al.
- Some countermeasures assume alternate access to at least two rows
- One-location hammering defeats countermeasures relying on this:
  - ANVIL
  - Corbet
• Some countermeasures assume that memory massaging requires a lot of memory
• Waylaying defeats countermeasures relying on this:
  • Gruss et al.
  • Van der Veen et al.
• Some countermeasures assume kernel data structure have to be targeted
• Opcode flipping defeats countermeasures relying on this:
  • G-CATT
# Opcode Flipping

<table>
<thead>
<tr>
<th>Bypass</th>
<th>Defense Class</th>
<th>Static Analysis</th>
<th>Performance Counters</th>
<th>Memory Access Pattern</th>
<th>Physical Proximity</th>
<th>Memory footprint</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Intel SGX</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td></td>
<td>One-location hammering</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td></td>
<td>Opcode flipping</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td></td>
<td>Memory waylaying</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>Defense class defeated</td>
<td></td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>
What do we learn from it?

- Many (academic) countermeasures were proposed to mitigate Rowhammer
What do we learn from it?

- Many (academic) countermeasures were proposed to mitigate Rowhammer
- We showed that all of them can be circumvented (Gruss, Lipp, et al. 2018)
What do we learn from it?

• Many (academic) countermeasures were proposed to mitigate Rowhammer
• We showed that all of them can be circumvented (Gruss, Lipp, et al. 2018)
• We cannot design countermeasures without completely understanding the attack
• Many (academic) countermeasures were proposed to mitigate Rowhammer
• We showed that all of them can be circumvented (Gruss, Lipp, et al. 2018)
• We cannot design countermeasures without completely understanding the attack
• Otherwise we only patch concrete exploits, but do not solve the problem
What do we learn from it?

• We have to invest more into researching attacks

There are still aspects of Rowhammer we do not fully understand. However, this is required to design effective countermeasures. Moreover, new features might introduce new attack vectors (e.g., SGX).
What do we learn from it?

• We have to invest more into researching attacks
• There are still aspects of Rowhammer we do not fully understand
What do we learn from it?

- We have to invest more into researching attacks
- There are still aspects of Rowhammer we do not fully understand
- However, this is required to design effective countermeasures
What do we learn from it?

• We have to invest more into researching attacks
• There are still aspects of Rowhammer we do not fully understand
• However, this is required to design effective countermeasures
• Moreover, new features might introduce new attack vectors (e.g., SGX)
What to we learn from it?

• We underestimated side-channel attacks for a long time

Industry and customers have to reconsider priorities → focus more on security instead of performance

Reliability issues (Rowhammer) can have security impacts

More research is required to understand attacks to ultimately mitigate them
What to we learn from it?

- We underestimated side-channel attacks for a long time
- Industry and customers have to reconsider priorities → focus more on security instead of performance
What to we learn from it?

• We underestimated side-channel attacks for a long time
• Industry and customers have to reconsider priorities → focus more on security instead of performance
• Reliability issues (Rowhammer) can have security impacts
What to we learn from it?

- We underestimated side-channel attacks for a long time
- Industry and customers have to reconsider priorities → focus more on security instead of performance
- Reliability issues (Rowhammer) can have security impacts
- More research is required to understand attacks to ultimately mitigate them
• Rowhammer attacks have a huge attack potential
• One-location hammering showed us that previous assumptions on how to trigger the Rowhammer bug are invalid
  • Keeping only one DRAM row open is sufficient
• Op-code flipping (in the sudo binary) allows to obtain root privileges
• Proposed countermeasures in software are ineffective
Rowhammer: From the Basics to Sophisticated New Variants

Moritz Lipp, Michael Schwarz

February 20, 2018

Graz University of Technology