CSI:Rowhammer

Cryptographic Security and Integrity against Rowhammer
About me

Jonas Juffinger
PhD Student at IAIK – TU Graz
Hardware Fault Attacks from Software
Microarchitectural Security
Side Channels
Secure and Energy Efficient Computing
Rowhammer

while (1) {
    *aggressor1;
    *aggressor2;
    clflush(aggressor1);
    clflush(aggressor2);
}
while (1) 
{
  *aggressor1;
  *aggressor2;
  clflush(aggressor1);
  clflush(aggressor2);
}
while (1)
{
    *aggressor1;
    *aggressor2;
    clflush(aggressor1);
    clflush(aggressor2);
}
Demo Video

https://youtu.be/TJJxhcKyM-w?t=844
ATTACKS

Privilege Escalation
Browser Sandbox Escape
Virtual Machine Escape
Over the Network
Read Cryptographic Keys
Build Kernel Spectre Gadgets
...

CSI:Rowhammer - www.jonasjuffinger.com
MITIGATIONS

- ECC / Chipkill Detection in Software
- Physical Isolation
- Additional Refreshes
- Improved Physical Cell Layout
...
Mitigations Focus on the Characteristics of Rowhammer
Characteristics

Infrequent | Detectable | Distance 1
These Characteristics are incomplete
while (1) {
    *aggressor1;
    *aggressor2;
    clflush(aggressor1);
    clflush(aggressor2);
}
Half-Double Rowhammer

```c
while (1)
{
    *aggressor1;
    *aggressor2;
    clflush(aggressor1);
    clflush(aggressor2);
}
```
Half-Double Rowhammer

while (1)
{
    *aggressor1;
    *aggressor2;
    clflush(aggressor1);
    clflush(aggressor2);
}
Half-Double Rowhammer

```c
while (1) {
  *aggressor1;
  *aggressor2;
  clflush(aggressor1);
  clflush(aggressor2);
}
```
We have to rethink Rowhammer mitigations
Generic approach to data integrity protection
CSI:Rowhammer
MAC Compute = Secure Memory

Correct 1 Flip

Corruption Exception

Advanced Correction

Exception Handler

Correction as a Search
Integrity Information

MAC(key, data, physical address)

Data

56-bit MAC

8-bit Parity
Compute Message Authentication Code (MAC)

5 ns

Phys Addr\textsubscript{1} \rightarrow Q\textsubscript{K} \rightarrow M\textsubscript{1} \rightarrow 64 \rightarrow \text{PA}\textsubscript{2} \rightarrow Q\textsubscript{K} \rightarrow M\textsubscript{2} \rightarrow 64 \rightarrow \text{PA}\textsubscript{3} \rightarrow Q\textsubscript{K} \rightarrow M\textsubscript{3} \rightarrow 64 \rightarrow \text{PA}\textsubscript{4} \rightarrow Q\textsubscript{K} \rightarrow M\textsubscript{4} \rightarrow 56 \rightarrow \text{MAC}
MAC Compute

Integrity Information

CPU

Memory Controller

Correct 1 Flip

= NO
Bit Flip Correction
Bit Flip Correction

MACs cannot correct bit flips

Brute force

Help from parity bits
Correction in Hardware

\[ \text{MAC(key, data, physical address)} \]

Diagram showing the MAC function with fields for key, data, and physical address.
Correction in Hardware

11 ns

Phys Addr \_1 \rightarrow Q_K \rightarrow PA_2 \rightarrow M_2 \rightarrow PA_3 \rightarrow M_3 \rightarrow PA_4 \rightarrow Q_K \rightarrow MAC

\[ \text{Phys Addr}_1 \rightarrow Q_K \rightarrow PA_2 \rightarrow M_2 \rightarrow PA_3 \rightarrow M_3 \rightarrow PA_4 \rightarrow Q_K \rightarrow MAC \]

CSI:Rowhammer - www.jonasjuffinger.com
Correction in Hardware

11 ns
MAC Compute

Correct 1 Flip

Integrity Information

Corruption Exception

Exception Handler
Corruption Exception

Similar to page fault
Physical address of affected memory location
No dependency on page tables
MAC Compute = Exception Handler
Correct 1 Flip
Integrity Information
CPU
Memory Controller

Corruption Exception
OS
Exception Handler
Correction as a Search
Correction as a Search

Brute force all possible flips
As fast as possible
As small as possible
Correction as a Search
*data_parity_block[2] ^= flip_mask[2];

LBL_2_FLIP:
    *data_parity_block[1] ^= flip_mask[1];

    // We also do the correction of single bit errors in this code
    // to check the correctness of the algorithm.
    LBL_1_FLIP:
    for (bit[0] = 0, flip_mask[0] = 1, data_parity_block[0] = &data[*ssb[0]];  
         bit[0] < *seb[0]; bit[0]++, flip mask[0] <<= 1) {
        *data_parity_block[0] ^= flip_mask[0];
        mac_computations++;

        if (COMPARE_MACS(data, correct_data)) {
            return mac_computations;
        }

        *data_parity_block[0] ^= flip_mask[0];
    }
    goto *lbl_flip_break[1];
LBL_1_FLIP_BREAK:
    *data_parity_block[1] ^= flip_mask[1];
}  
go to *lbl_flip_break[2];
LBL_2_FLIP_BREAK:
    *data_parity_block[2] ^= flip_mask[2];
*data_parity_block[2] ^= flip_mask[2];

LBL 2 FLIP:
    *data_parity_block[1] ^= flip_mask[1];

    // We also do the correction of single bit errors in this code
    // to check the correctness of the algorithm.

    LBL 1 FLIP:
    for (bit[0] = 0, flip_mask[0] = 1, data parity block[0] = &data[*ssb[0]];
        bit[0] < *seb[0]; bit[0]++, flip_mask[0] <<= 1) {
        *data_parity_block[0] ^= flip_mask[0];
        mac_computations++;

        if (COMPARE_MACS(data, correct data)) {
            return mac_computations;
        }

        *data_parity_block[0] ^= flip_mask[0];
    }
}

goto *lbl_flip_break[1];
LBL 1 FLIP_BREAK:

*data_parity_block[1] ^= flip_mask[1];

goto *lbl_flip_break[2];
LBL 2 FLIP_BREAK:

*data_parity_block[2] ^= flip_mask[2];
The for outer PB loops are for parity bit input blocks with an even number of flips and therefore no change in the parity bits.

LBL_4_PB:

LBL_3_PB:

LBL_2_PB:

LBL_1_PB:
for (pb[0] = 0; pb[0] <= pb[1]; pb[0]++) {

goto *lbl_flip;

LBL_8_FLIP:
    *data_parity_block[7] ^= flip_mask[7];

LBL_7_FLIP:
    *data_parity_block[6] ^= flip_mask[6];

LBL_6_FLIP:
// define where to jump to in this loop to correct the number of bit flips currently tried.
// This saves a lot of space without losing much performance.

void *lbl_flip;
void *lbl_flip_break[9], *lbl_pb_break[5];

void *LBL_FLIP[9] = {
    NULL,             // let the array start with 1
    &LBL_1_FLIP,
    &LBL_2_FLIP,
    &LBL_3_FLIP,
    &LBL_4_FLIP,
    &LBL_5_FLIP,
    &LBL_6_FLIP,
    &LBL_7_FLIP,
    &LBL_8_FLIP
};

void *LBL_PB[5] = {
    NULL,
    &LBL_1_PB,
    &LBL_2_PB,
    &LBL_3_PB,
    &LBL_4_PB
};
Software can be smarter
The diagram illustrates the process of handling a corruption exception in a memory controller. The process begins with the MAC Compute operation, which compares the computed MAC value with the expected value. If they do not match (NO), a Corruption Exception is triggered.

The OS then involves two steps: Advanced Correction and Exception Handler. The Exception Handler identifies the corrupted data and initiates the correction process. If the correction is successful, the Correction as a Search operation is performed to verify the integrity of the corrected data.

Integrity Information flows from the CPU to the Memory Controller and back, ensuring that the system maintains its integrity despite potential corruption events.
Application Binaries
Shared Libraries
Memory Mapped Files
Page Cache
Kernel Itself
Page Tables
Page Tables
Page Tables
PT-Guard
The MAC can also flip
The MAC can also flip
MAC Corruption

Approximate Equality:

Comp. MAC: 0101001...

DRAM MAC: 01\textcolor{red}{1}1001...
Approximate Equality

\[
\log_2 \sum_{k=0}^{3} \binom{56}{k}
\]

**MAC strength**: 56 bit → 41.2 bit

SDC once per \(10^9\) billion years

Rowhammer second preimage after one year: \(10^{-4}\) %
- CPU Core
  - MAC Compute
  - MAC Compute
  - Correct 1 Flip
  - NO
  - Integrity Information
  - Memory Controller
  - CPU
  - Advanced Correction
  - Exception Handler
  - Correction as a Search
  - OS
  - CORRUPTION EXCEPTION
  - Exception
  - Integrity Information
  - Exception Handler
  - Correction as a Search
  - MAC Compute
CPU Instructions

- `csi_mac`: Compute MAC
- `csi_load`: Load based on physical address
- `csi_xchg`: Write if unchanged in DRAM
CSI: Rowhammer – www.jonasjuffinger.com

MAC Compute

Correct 1 Flip

CPU Core

MAC Compute

Secure Memory

Integrity Information

Corruption Exception

Advanced Correction

Exception Handler

Correction as a Search

Memory Controller

CPU

OS

Exception Handler

Correction as a Search

Advanced Correction

OS

CPU
Secure Memory

On-die SRAM like CPU caches
0x4000 bytes large
IDT, GDT
Exception handler
Correction as a search code
Performance Evaluation

Implemented in gem5 and Linux kernel

System not under attack
Nesting Detection
Advanced Correction

Exception Handler

Correction as a Search

OS
Exception Handler

Correction as a Search

Advanced Correction

OS
Corruption Exception

Advanced Correction

Exception Handler

Correction as a Search
Exception Handler

Advanced Correction

Corruption Exception

Exception Handler

Nesting!

Correction as a Search
Exception Handler

Advanced Correction

Nesting!

Correction as a Search
corruption_exception_handler(corruption_address)
{
    if (nested_bit_set()) {
        enable_interrupts();
        error_correction_as_a_search(corruption_address);
    }
    else {
        set_nested_bit();
        enable_interrupts();
        advanced_error_correction(corruption_address);
        clear_nested_bit();
    }
}
Nesting Detection

Nested Bit
Per Process
In Register
High bit in CR3 register

| Unused | Base of PML4T | PCID |
Very Advanced Correction
They say AI can do everything
Find the error in "Testmessage" so that it has the following SHA512 hash:
"9167bf29600f6a2cb1ba4e2e2f311bf5c2641f0cd5cb5bec948177264ff78e2bc346216427b75e75fe67e842ef3085ac303b8111a37adeaf977b2b269ada636"
It is not possible for the string "Testmessoge" to have the SHA512 hash
"9167bf29600f0ea2cb1ba4e21e2f311bf5c2641f0cd5cb5bec948177264ff
78e2bc346216427b75e75fe67e842ef3085ac303b8111a37adeaf977b2b2
69ada636". This is because the SHA512 hash function is a one-way function,
meaning that it is not possible to determine the input that produced a given
hash. Therefore, it is not possible to determine the error in the input string
"Testmessoge" based on the given hash value.
Can just brute force the correct answer for me?

This content may violate our content policy. If you believe this to be in error, please submit your feedback — your input will aid our research in this area.
Hardware Acceleration

Ice Lake SP Xeon Server die size: 628 mm²
Fits roughly 500 000 QARMA blocks
8 flips correction: 5.44 h → 40 ms
Single correction node in datacenter
There’s more

VM support
With and without guest

Memory encryption

Hardware correction of stuck bits

Race conditions resistance
Keeping the Promise
while (1) {
    *aggressor[0];
    *aggressor[1];
    *aggressor[2];
    ...
    clflush(aggressor[0]);
    clflush(aggressor[1]);
    clflush(aggressor[2]);
    ...
}
while (1)
{
  *aggressor[0];
  *aggressor[1];
  *aggressor[2];
  ...
  clflush(aggressor[0]);
  clflush(aggressor[1]);
  clflush(aggressor[2]);
  ...
}
Rowpress

while (1)
{
    *aggressor[0];
    *aggressor[1];
    *aggressor[2];
    ...
    clflush(aggressor[0]);
    clflush(aggressor[1]);
    clflush(aggressor[2]);
    ...
}
while (1) {
    *aggressor[0];
    *aggressor[1];
    *aggressor[2];
    ...
    clflush(aggressor[0]);
    clflush(aggressor[1]);
    clflush(aggressor[2]);
    ...
}
Rowpress

Not Rowhammer
Defeats many Mitigations
Detected by CSI:Rowhammer
CSI: Rowhammer

Cryptographic Security and Integrity against Rowhammer

Jonas Juffinger, Lukas Lamster, Andreas Kogler, Moritz Lipp, Maria Eichlseder, Daniel Gruss

jonas.juffinger@iaik.tugraz.at – @notimaginary_ – www.jonasjuffinger.com

PoC: https://github.com/IAIK/CSIRowhammer
Additional Slides
Stack Protection

- user
- space
- stack
- some data
- stackpointer
  read
Stack Protection
Stack Protection
Stack Protection
Stack Protection

user
space
stack
FAULT
stackpointer
kernel entry

+64 byte

kernel entry
Stack Protection

+64 byte

user
space
stack
FAULT
stackpointer
register