Repurposing Segmentation as a Practical LVI-NULL Mitigation in SGX

Lukas Giner, Andreas Kogler, Claudio Canella, Michael Schwarz, Daniel Gruss
May 06, 2022
Who am I?

Lukas Giner
PhD student @ Graz University of Technology

@redrabbyte
lukas.giner@iaik.tugraz.at
Motivation

<table>
<thead>
<tr>
<th>Method</th>
<th>Overhead [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMERIC SORT</td>
<td>128.13</td>
</tr>
<tr>
<td>BITFIELD</td>
<td>6.29 \cdot 10^{-4}</td>
</tr>
<tr>
<td>FP EMULATION</td>
<td>1221.08</td>
</tr>
<tr>
<td>FOURIER</td>
<td>133.19</td>
</tr>
<tr>
<td>ASSIGNMENT</td>
<td>684.05</td>
</tr>
<tr>
<td>IDEA</td>
<td>221.9</td>
</tr>
<tr>
<td>HUFFMAN</td>
<td>303.46</td>
</tr>
<tr>
<td>NEURAL NET</td>
<td>72.53</td>
</tr>
<tr>
<td>LU DECOMPOSITION</td>
<td>132.45</td>
</tr>
<tr>
<td>STRING SORT</td>
<td>5</td>
</tr>
</tbody>
</table>

Lukas Giner (@redrabbyte)
- CPU does many things, all at once
• CPU does many things, all at once
• Parallelizing instructions speeds up execution
CPU does many things, all at once
Parallelizing instructions speeds up execution
Mistakes?
• CPU does many things, all at once
• Parallelizing instructions speeds up execution
• Mistakes?
• Roll back to the mistake, either raise an error or try again
CPU does many things, all at once
Parallelizing instructions speeds up execution
Mistakes?
Roll back to the mistake, either raise an error or try again
Undone instructions are called transient
• What can an attacker do with transient instructions?
• What can an attacker do with transient instructions?
• Not all state is gone - traces in μArch state
Transient Execution 2 - Attacker’s Perspective

• What can an attacker do with transient instructions?
• Not all state is gone - traces in \( \mu \text{Arch state} \)
• Caching!
User Memory

\[
\begin{array}{cccccc}
A & B & C & D & E & F \\
G & H & I & J & K & L \\
M & N & O & P & Q & R \\
S & T & U & V & W & X \\
Y & Z & & & & \\
\end{array}
\]

\[\text{char value} = \text{kernel}[0]\]
User Memory

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>O</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{char value} = \text{kernel}[0]
\]

Page fault (Exception)
Meltdown: Transiently encoding unauthorized memory

char value = kernel[0]
mem[value]

Page fault (Exception)
Out of order

User Memory

A  B
C  D  E
F  G  H
I  J  K
L  M  N
O  P  Q
R  S  T
U  V  W
X  Y  Z
Meltdown: Transiently encoding unauthorized memory

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>E</td>
<td>F</td>
</tr>
<tr>
<td>G</td>
<td>H</td>
</tr>
<tr>
<td>I</td>
<td>J</td>
</tr>
<tr>
<td>L</td>
<td>M</td>
</tr>
<tr>
<td>O</td>
<td>P</td>
</tr>
<tr>
<td>R</td>
<td>S</td>
</tr>
<tr>
<td>U</td>
<td>V</td>
</tr>
<tr>
<td>X</td>
<td>Y</td>
</tr>
</tbody>
</table>

```
char value = kernel[0]
```

mem[value]

Page fault (Exception)

Out of order

Lukas Giner (@redrabbyte)
• What if we turn Meltdown around?
LVI - Turning Meltdown around

- What if we turn Meltdown around?
- Meltdown
What if we turn Meltdown around?
Instead of leaking, we insert a value
LVI - Turning Meltdown around

• What if we turn Meltdown around?
• Instead of leaking, we insert a value
• Transiently steer the victim to give up data!
• What if we turn Meltdown around?
• Instead of leaking, we insert a value
• Transiently steer the victim to give up data!
• Exfiltrate with cache, as usual
• How do these values get there?
• How do these values get there?
• No need to care, Intel fixed it!
• How do these values get there?
• No need to care, Intel fixed it!
  (it’s buffers and loose address matching)
• How do these values get there?
• No need to care, Intel fixed it!
  (it’s buffers and loose address matching)
• New CPUs just “inject” 0 instead
• How do these values get there?
• No need to care, Intel fixed it!
  (it’s buffers and loose address matching)
• New CPUs just “inject” 0 instead
• Problem solved?
LVI-NULL Vectors

**Control Flow**

- **Direct Jump**
  - `mov (mem),%reg`
  - `call *%reg`
  - Execute page 0 e.g. via vtable
  - `- r(w)x page 0`
  - `- if non-writeable: gadget`

- **Indirect Jump**
  - `mov (mem),%reg`
  - `call *(%reg)`
  - Run address at 0, e.g. vtable
  - `- r(w) page 0`
  - `- if non-writeable: gadget`

- **Transient Stack**
  - `pop esp`
  - `ret`
  - ROP via stack on page 0
  - `- rw page 0`

- **Branches**
  - `cmp %reg,(mem)`
  - `je *offs`
  - Change branch target, special case: switch
  - `- gadget`

**Data Flow**

- **Direct Load**
  - `mov (mem),%reg`
  - Injects null e.g. faulting aes
  - `- gadget`

- **Indirect Load**
  - `mov (mem),%reg`
  - Injects arbitrary values
  - `- r(w) page 0`
LVI-NULL Vectors 2

Example asm sequence:
- `mov (mem), reg`
- `call *(reg)`
- Run address at 0
  - E.g. vtable
- `r(w) page 0`
  - Gadget if non-writeable

Indirect load:
- `mov (mem), reg`
- `mov (reg), reg`
- Injects arbitrary values in code
- `r(w) page 0`

Indirect jump:
- `mov (mem), reg`
- `call *(reg)`
- Run address at 0
- E.g. vtable
- `r(w) page 0; gadget if non-writeable

Attacks scenario:
- Requirement:
  - `r(w) page 0; gadget if non-writeable`
• SGX protects enclave from OS
Application

- SGX protects enclave from OS

Untrusted part

Create Enclave

Operating System
- SGX protects enclave from OS

**Application**

- Untrusted part
  - Create Enclave

- Trusted part
  - Trusted Fnc.

**Operating System**

- SGX protects enclave from OS
- SGX protects enclave from OS
SGX Application

Untrusted part

Create Enclave

Call Trusted Fnc.

Trusted part

Call Gate

Trusted Fnc.

Operating System

- SGX protects enclave from OS
- SGX protects enclave from OS
• SGX protects enclave from OS
SGX protects enclave from OS.

- SGX protects enclave from OS.
SGX

Application

Untrusted part

- Create Enclave
- Call Trusted Fnc.

Trusted part

- Call Gate
- Trusted Fnc.
- Return

Operating System

- SGX protects enclave from OS
• SGX protects enclave from OS
SGX

Application

Untrusted part

Create Enclave

Call Trusted Fnc.

Call Gate

Trusted part

Trusted Fnc.

Return

Operating System

- SGX protects enclave from OS
- But OS still controls parts of the PTE
- SGX protects enclave from OS
- But OS still controls parts of the PTE
- Removing accessed bit injects faults
SGX protects enclave from OS
But OS still controls parts of the PTE
Removing accessed bit injects faults
Instruction-targeted injection with SGX-Step
Mitigating in Software

• Problem: Injecting 0 allows attacker controlled loads
Mitigating in Software

- Problem: Injecting 0 allows attacker controlled loads
- Can we prevent redirection to zero page?
Mitigating in Software

- Problem: Injecting 0 allows attacker controlled loads
- Can we prevent redirection to zero page?
- Make all loads relative to enclave memory!

Revive an ancient mechanism, Segmentation!

enclave image

enclave start

enclave end

GS-base

unreadable pages

Lukas Giner (@redrabbyte)
Mitigating in Software

- Problem: Injecting 0 allows attacker controlled loads
- Can we prevent redirection to zero page?
- Make all loads relative to enclave memory!
- Revive an ancient mechanism, Segmentation!
• Problem: Injecting 0 allows attacker controlled loads
• Can we prevent redirection to zero page?
• Make all loads relative to enclave memory!
• Revive an ancient mechanism, Segmentation!
<table>
<thead>
<tr>
<th>push %rbp</th>
<th>sub $0x8,%rsp</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov %rbp,%gs: (%rsp)</td>
<td></td>
</tr>
</tbody>
</table>

- All loads over gs
Mitigated Code

- All loads over gs
- Instruction with implicit load

```
push %rbp
sub $0x8,%rsp
mov %rbp,%gs: (%rsp)
```
Mitigated Code

- All loads over gs
- Instruction with implicit load → replace with other instructions

```assembly
push %rbp
sub $0x8,%rsp
mov %rbp,%gs: (%rsp)
```
Mitigated Code

- All loads over gs
- Instruction with implicit load → replace with other instructions
Mitigated Code

- All loads over gs
- Instruction with implicit load → replace with other instructions
Mitigated Code

- All loads over gs
- Instruction with implicit load → replace with other instructions

```assembly
push %rbp
sub $0x8,%rsp
mov %rbp,%gs(%rsp)
callq 400480 <func>
leaq $return_address(%rip),%r11
sub $0x8,%rsp
mov %r11,%gs(%rsp)
jmpq 400480 <func>
pop %rbp
mov %gs(%rsp),%rbp
add $0x8,%rsp
retq
mov %gs(%rsp),%rcx
add $0x8,%rsp
jmpq *%rcx
```
Overhead [%]

- NUMERIC SORT
- BITFIELD
- FP EMULATION
- FOURIER
- ASSIGNMENT
- IDEA
- HUFFMAN
- NEURAL NET
- LU DECOMPOSITION
- STRING SORT

LVI-NULLify, clang-lvi-cfi, clang-lvi-opt
Performance

<table>
<thead>
<tr>
<th>Test Case</th>
<th>LVI-NULLify</th>
<th>clang-lvi-cfi</th>
<th>clang-lvi-opt</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMERIC SORT</td>
<td>0.7%</td>
<td>0.21%</td>
<td>128.13%</td>
</tr>
<tr>
<td>BITFIELD</td>
<td>0.93%</td>
<td>0.33%</td>
<td>6.29 \times 10^{-4}%</td>
</tr>
<tr>
<td>FP EMULATION</td>
<td>4.87%</td>
<td>30.48%</td>
<td>30.48%</td>
</tr>
<tr>
<td>FOURIER</td>
<td>8.93%</td>
<td>80.97%</td>
<td>133.19%</td>
</tr>
<tr>
<td>ASSIGNMENT</td>
<td>3.28%</td>
<td>3.36%</td>
<td>2.38%</td>
</tr>
<tr>
<td>IDEA</td>
<td>2.38%</td>
<td>13.05%</td>
<td>221.9%</td>
</tr>
<tr>
<td>HUFFMAN</td>
<td>9.55%</td>
<td>0.24%</td>
<td>303.46%</td>
</tr>
<tr>
<td>NEURAL NET</td>
<td>3.47%</td>
<td>57.58%</td>
<td>53.63%</td>
</tr>
<tr>
<td>LU DECOMPOSITION</td>
<td>4.32%</td>
<td>53.63%</td>
<td>132.45%</td>
</tr>
<tr>
<td>STRING SORT</td>
<td>0.65%</td>
<td>1.49%</td>
<td>4.32%</td>
</tr>
</tbody>
</table>

Lukas Giner (@redrabbyte)
Conclusion

- LVI-NULL is still here (in Rocket Lake)...

LVI-NULLify on GitHub
LVI-NULL is still here (in Rocket Lake)...

..but SGX isn't, in affected models
Conclusion

- LVI-NULL is still here (in Rocket Lake)...
  ..but SGX isn't, in affected models
- For now, only Comet Lake is affected
Conclusion

• LVI-NULL is still here (in Rocket Lake)...
  ..but SGX isn't, in affected models
• For now, only Comet Lake is affected
• Maybe relative addressing will be useful somewhere else?
Repurposing Segmentation as a Practical LVI-NULL Mitigation in SGX

Lukas Giner (@redrabbyte), Andreas Kogler (@0xhilbert), Claudio Canella (@cc0x1f)
Michael Schwarz (@misc0110), Daniel Gruss (@lavados)