LVI

Hijacking Transient Execution through Microarchitectural Load Value Injection

Jo Van Bulck [1] **Daniel Moghimi** [2] **Michael Schwarz** [3] **Moritz Lipp** [3] **Marina Minkin** [4]
Daniel Genkin [4] **Yuval Yarom** [5] **Berk Sunar** [2] **Daniel Gruss** [3] **Frank Piessens** [1]

41st IEEE Symposium on Security and Privacy (digital edition) – May 18, 2020

[1]imec-DistriNet, KU Leuven [2]Worcester Polytechnic Institute [3]Graz University of Technology
[4]University of Michigan [5]University of Adelaide and Data61

Hijacking transient execution?

Spectre

Meltdown

Hijacking transient execution?

Spectre

v1, v2, v4, v5,
Spectre-BTB,
Spectre-RSB,
ret2spec,
SGXPectre,
SmotherSpectre,
NetSpectre?

Meltdown

v3, v3.1, v3a,
RDCL?

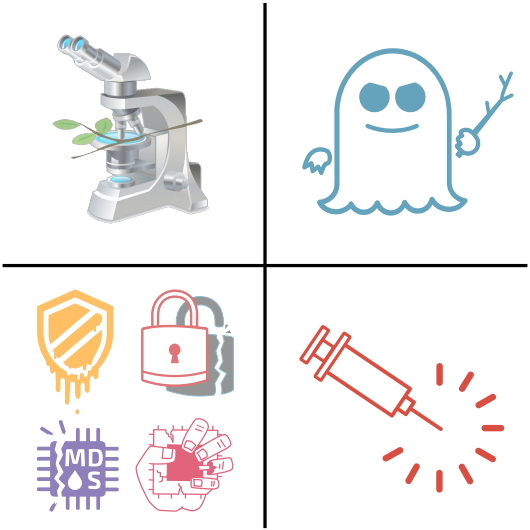
ZombieLoad, MDS?

Foreshadow
Foreshadow-NG,
L1TF?

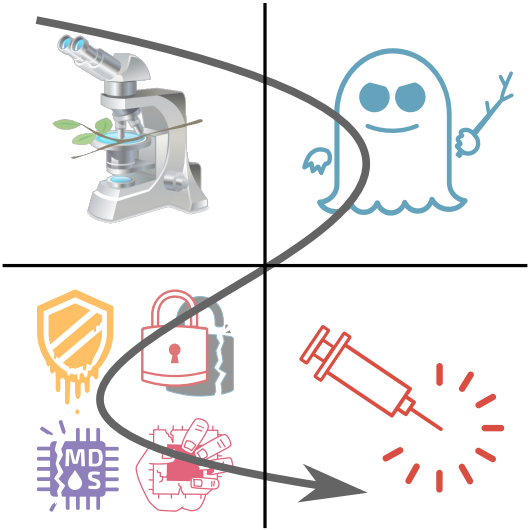
RIDL, Fallout?


Welcome to
London Bridge

A new perspective. . .

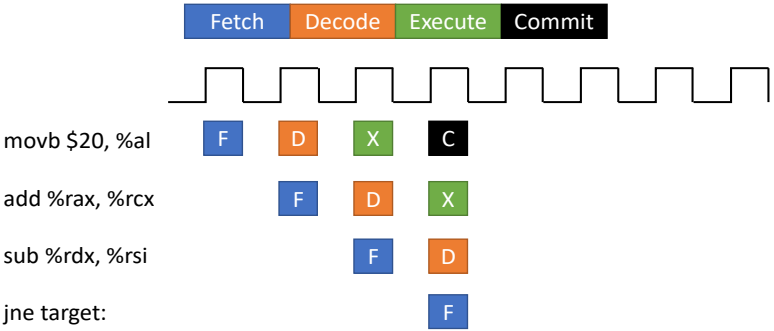


A new perspective. . .

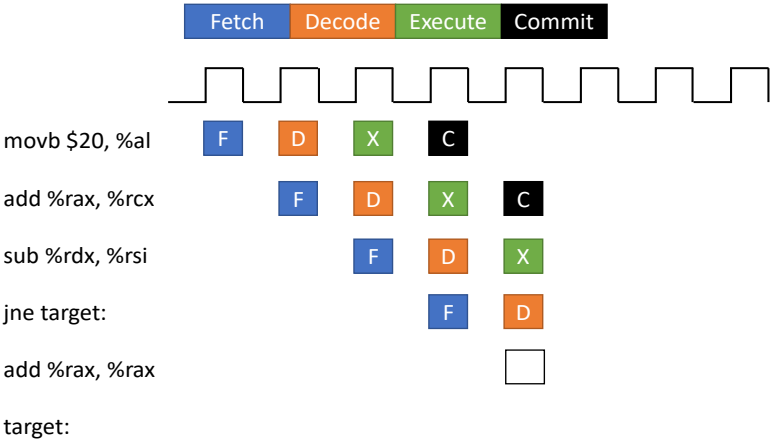



BACK
TO THE FUTURE
OUTATIME

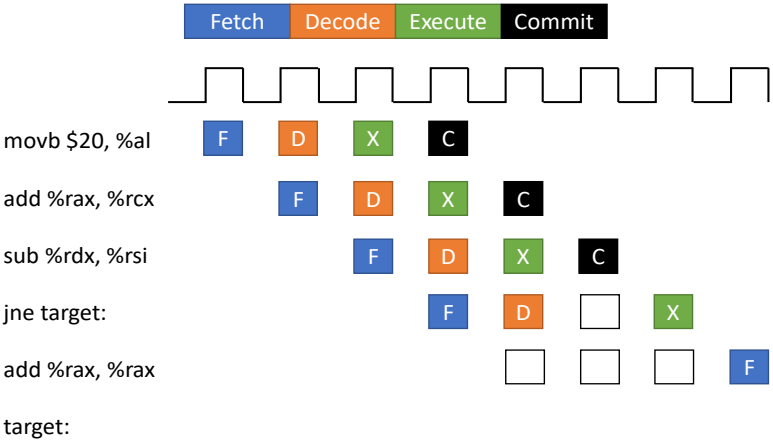
Motivation: Branch prediction to avoid pipeline stalls



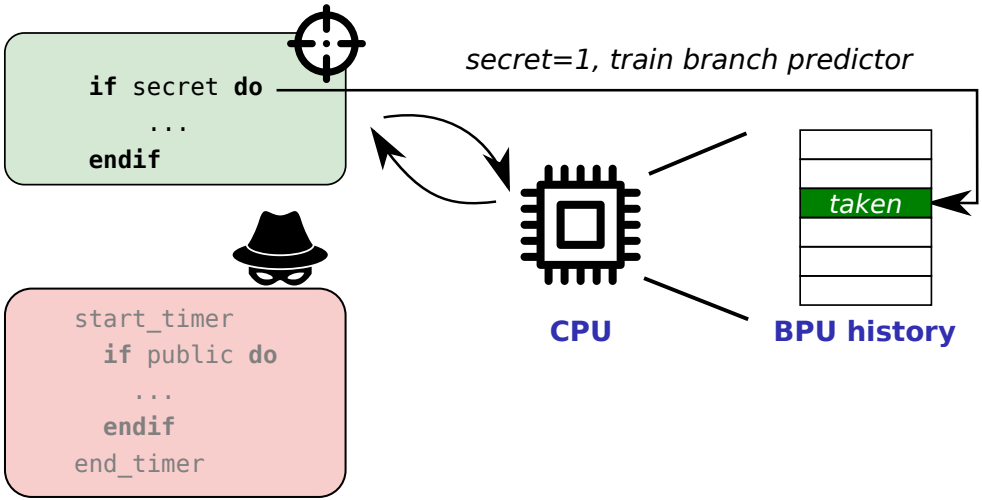
Motivation: Branch prediction to avoid pipeline stalls



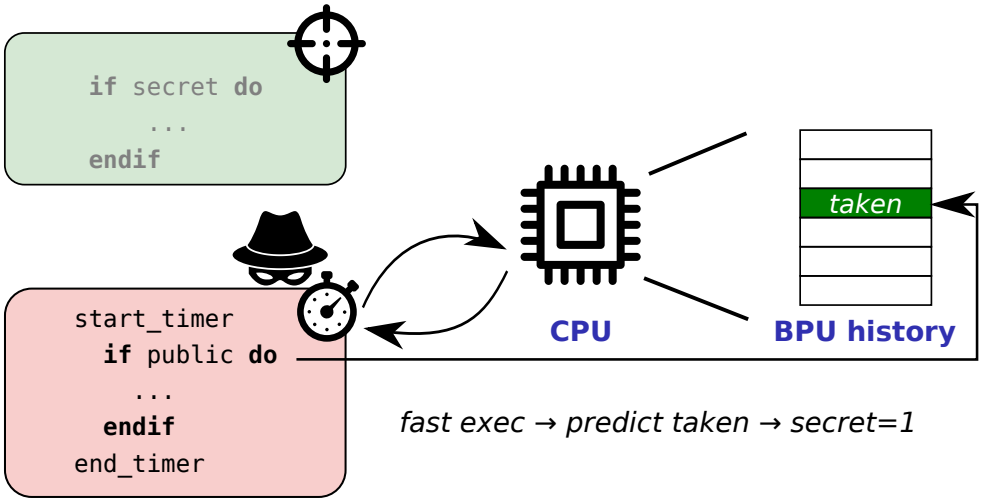
Motivation: Branch prediction to avoid pipeline stalls

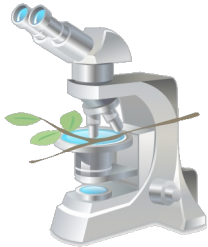


Branch prediction as a side channel



Branch prediction as a side channel

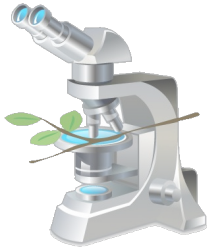


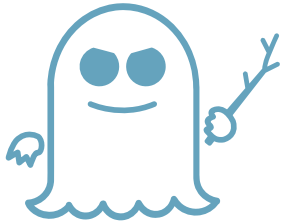


?

?

?



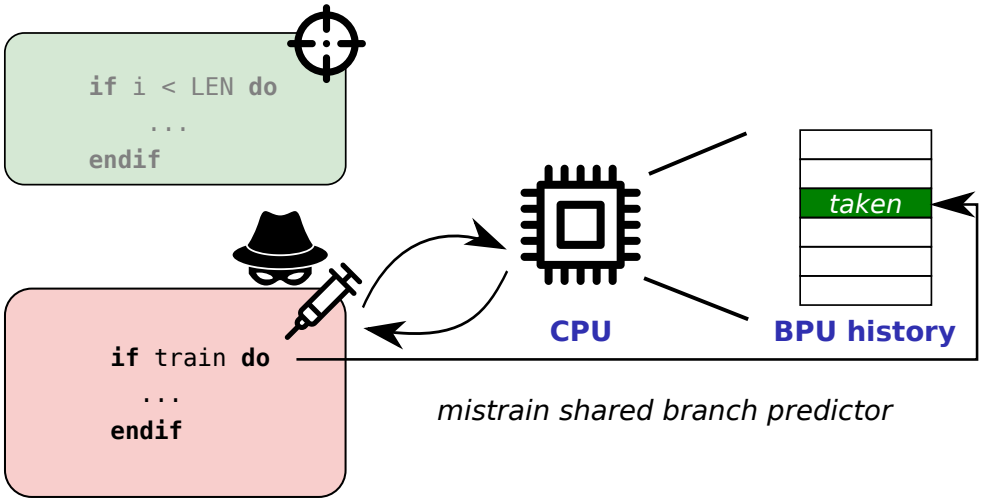


?

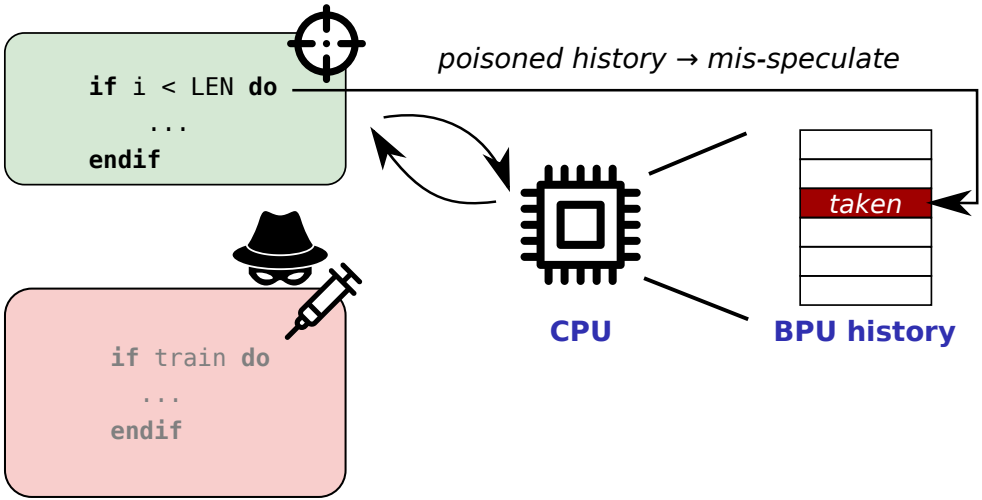
?


THE WHITE HOUSE
6:14 PM
WHITE HOUSE
WASHINGTON
BREAKING NEWS
PRES. TRUMP UPDATES PUBLIC ON FEDERAL RESPONSE TO VIRUS
MSNBC

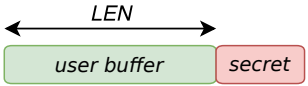
Branch prediction as a transient fault injection primitive



Branch prediction as a transient fault injection primitive



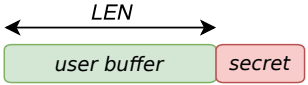
Spectre v1: Speculative buffer over-read




```
if (idx < LEN)
{
  s = buffer[idx];
  t = lookup[s];
  ...
}
```

- Programmer *intention:* no out-of-bounds accesses

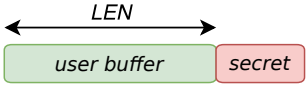
Spectre v1: Speculative buffer over-read




```
if (idx < LEN)
{
  s = buffer[idx];
  t = lookup[s];
  ...
}
```

- Programmer *intention:* no out-of-bounds accesses
- **Mistrain gadget** to speculatively "ahead of time" execute with $idx \geq LEN$ in the transient world

Spectre v1: Speculative buffer over-read

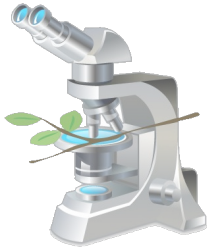


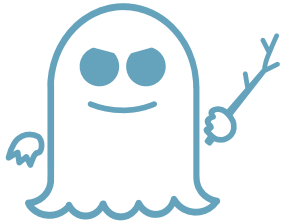

```
if (idx < LEN)
{
  s = buffer[idx];
  t = lookup[s];
  ...
}
```

- Programmer *intention:* no out-of-bounds accesses
- **Mistrain gadget** to speculatively "ahead of time" execute with $idx \geq LEN$ in the transient world
- **Side channels** may leave traces after roll-back!

Spectre take-away

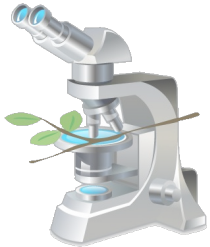
- CPU **transiently** executes wrong code paths
- **Confused-deputy gadgets** encode secrets via side channels

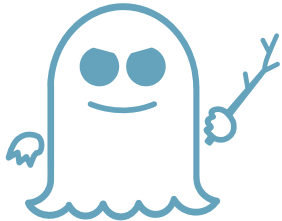




?

?

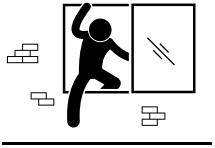





MD
S

?

Meltdown: Transiently encoding unauthorized memory



Unauthorized access

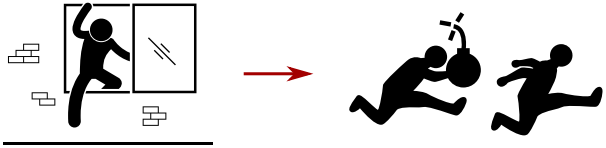
Listing 1: x86 assembly

```
1 meltdown:
2   // %rdi: oracle
3   // %rsi: secret_ptr
4
5   movb (%rsi), %al
6   shl $0xc, %rax
7   movq (%rdi, %rax), %rdi
8   retq
```

Listing 2: C code.

```
1 void meltdown(
2       uint8_t *oracle,
3       uint8_t *secret_ptr)
4 {
5   uint8_t v = *secret_ptr;
6   v = v * 0x1000;
7   uint64_t o = oracle[v];
8 }
```

Meltdown: Transiently encoding unauthorized memory



Unauthorized access

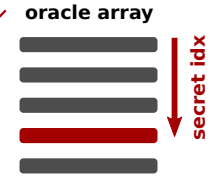
Transient out-of-order window

Listing 1: x86 assembly.

```
1 meltdown:
2     // %rdi: oracle
3     // %rsi: secret_ptr
4
5     movb (%rsi), %al
6     shl $0xc, %rax
7     movq (%rdi, %rax), %rdi
8     retq
```

Listing 2: C code.

```
1 void meltdown(
2         uint8_t *oracle,
3         uint8_t *secret_ptr)
4 {
5     uint8_t v = *secret_ptr;
6     v = v * 0x1000;
7     uint64_t o = oracle[v];
8 }
```



Meltdown: Transiently encoding unauthorized memory

Unauthorized access

Transient out-of-order window

Exception
(discard architectural state)

Listing 1: x86 assembly.

```
1 meltdown:
2     // %rdi: oracle
3     // %rsi: secret_ptr
4
5     movb (%rsi), %al
6     shl $0xc, %rax
7     movq (%rdi, %rax), %rdi
8     retq
```

Listing 2: C code.

```
1 void meltdown(
2         uint8_t *oracle,
3         uint8_t *secret_ptr)
4 {
5     uint8_t v = *secret_ptr;
6     v = v * 0x1000;
7     uint64_t o = oracle[v];
8 }
```

Meltdown: Transiently encoding unauthorized memory

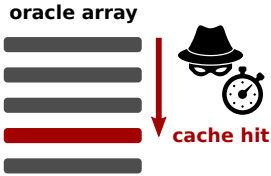
Unauthorized access | Transient out-of-order window | **Exception handler**

Listing 1: x86 assembly.

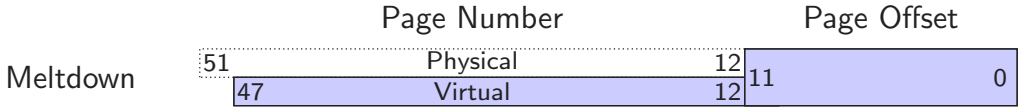
```
1 meltdown:
2   // %rdi: oracle
3   // %rsi: secret_ptr
4
5   movb (%rsi), %al
6   shl $0xc, %rax
7   movq (%rdi, %rax), %rdi
8   retq
```

Listing 2: C code.

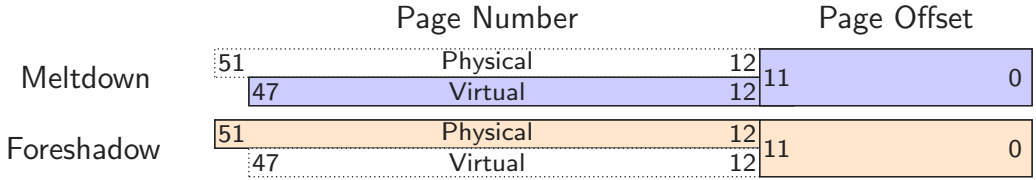
```
1 void meltdown(
2       uint8_t *oracle,
3       uint8_t *secret_ptr)
4 {
5   uint8_t v = *secret_ptr;
6   v = v * 0x1000;
7   uint64_t o = oracle[v];
8 }
```



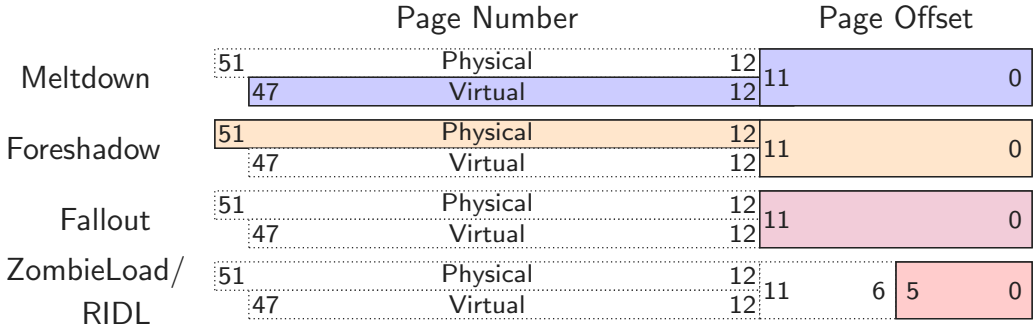
Meltdown variants: Address dependencies



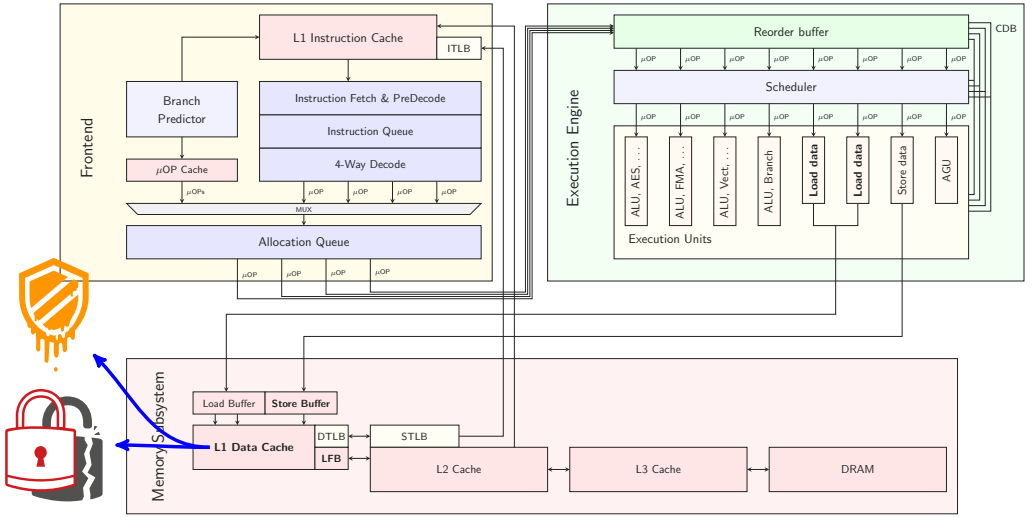
Meltdown variants: Address dependencies



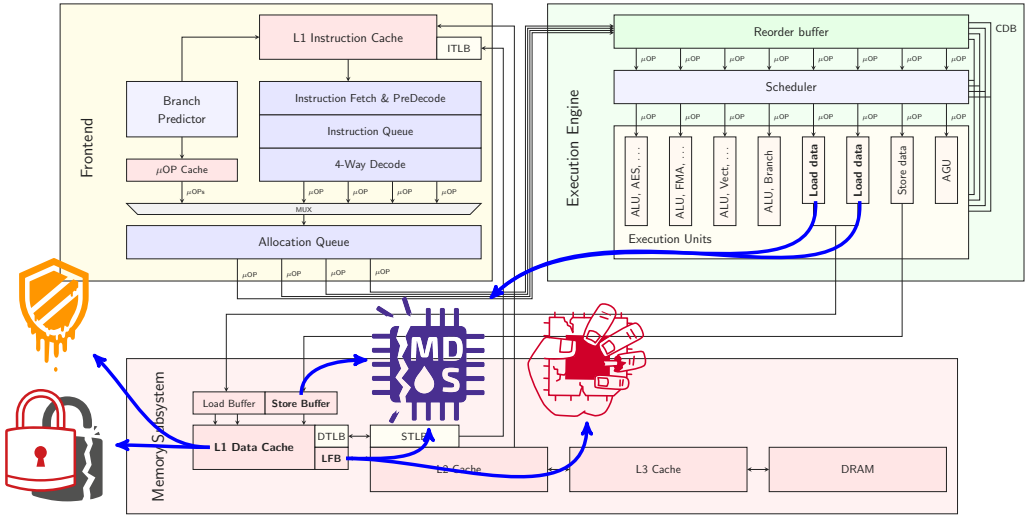
Meltdown variants: Address dependencies



Meltdown variants: Microarchitectural buffers

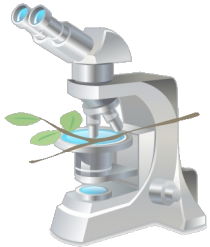


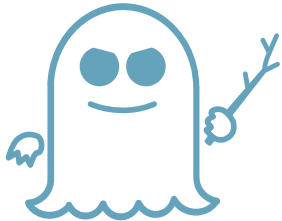
Meltdown variants: Microarchitectural buffers



Meltdown take-away

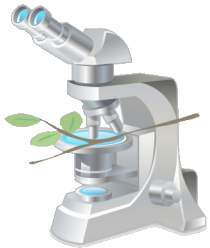
Faulting (or assisted) loads transiently forward **unrelated data** from various microarchitectural buffers

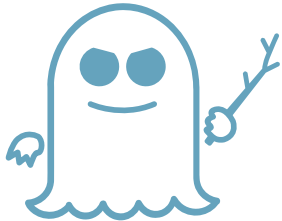




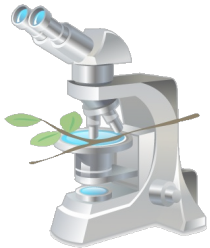

MD
S

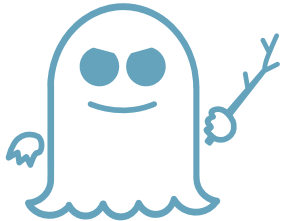
?



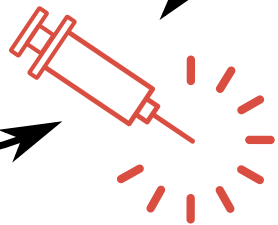



MD
S

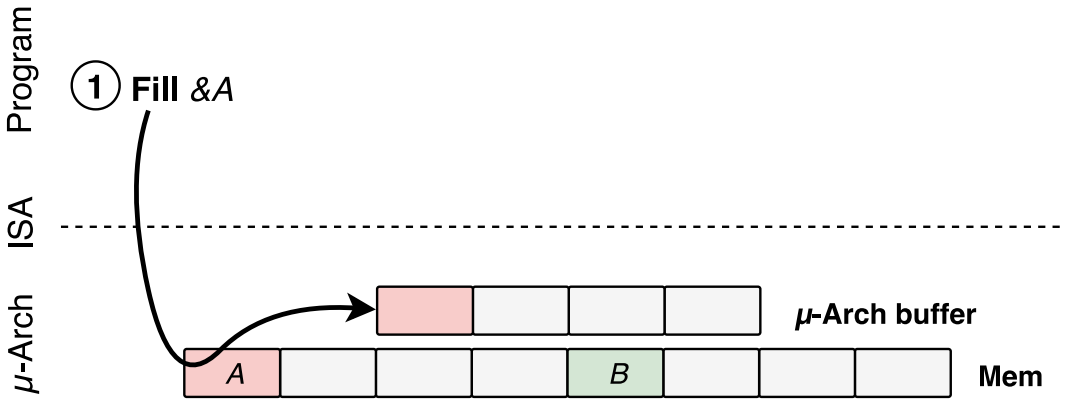




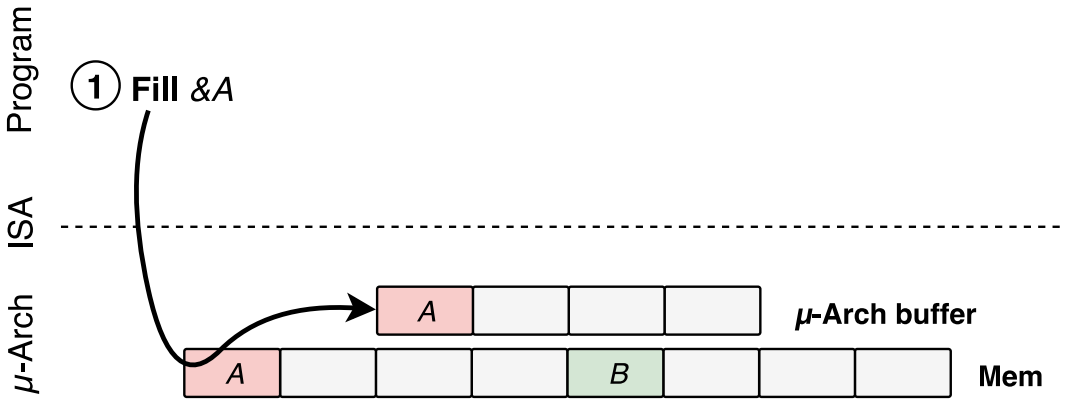

MD
S



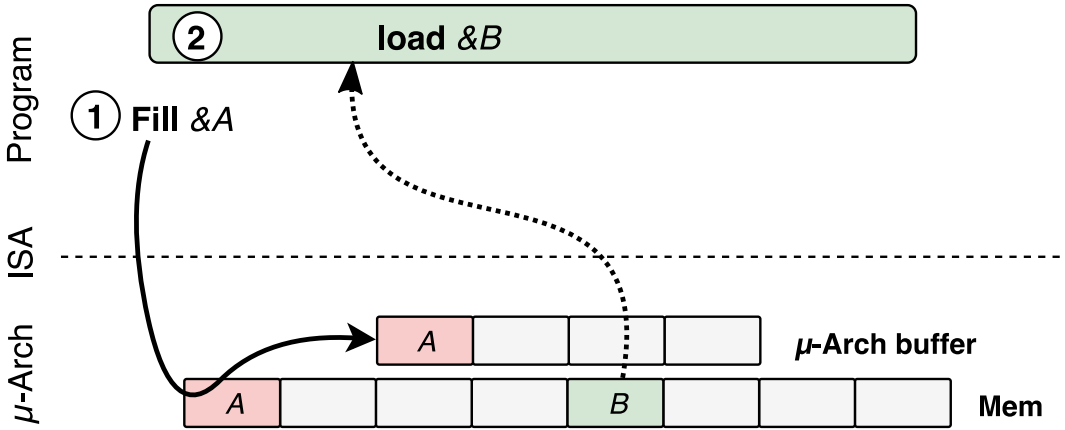
LVI: The basic idea



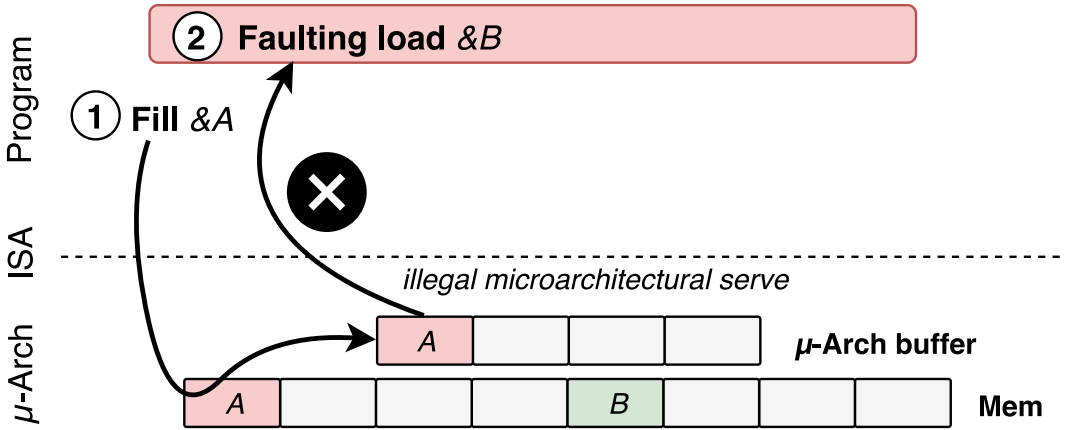
LVI: The basic idea



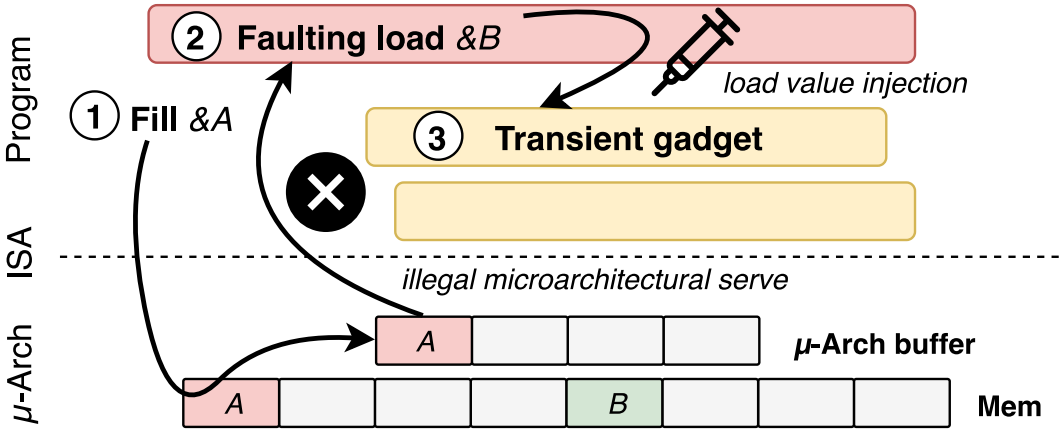
LVI: The basic idea



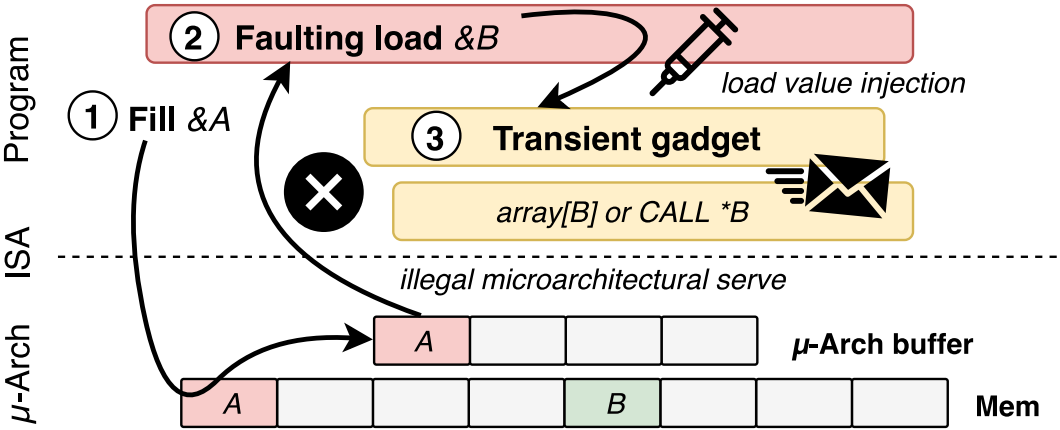
LVI: The basic idea



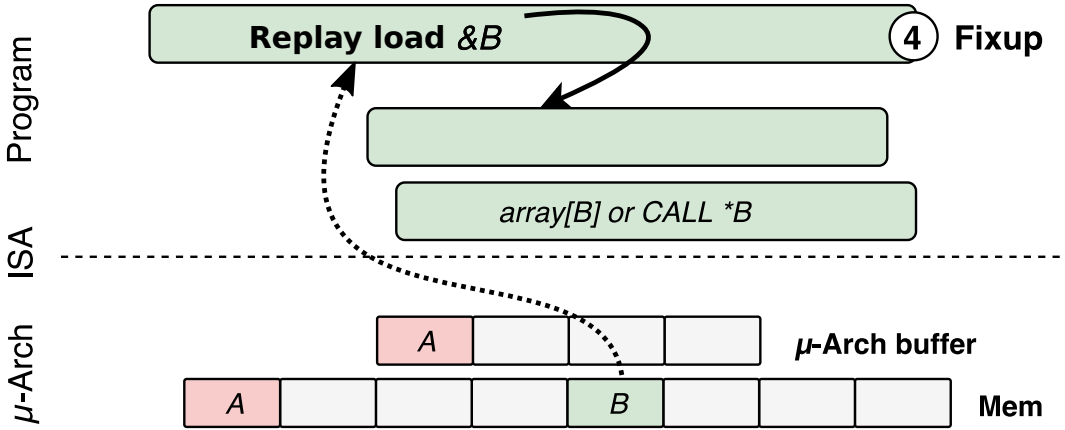
LVI: The basic idea



LVI: The basic idea

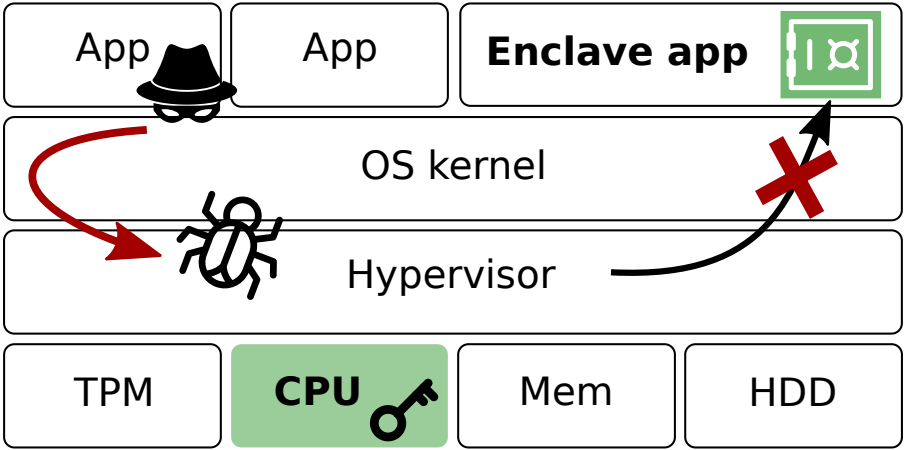


LVI: The basic idea



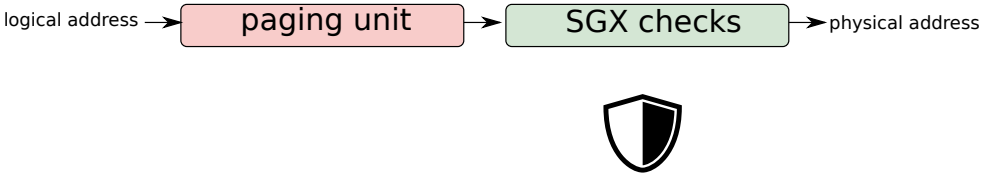

BUT WAIT...
A PAGE FAULT IN THE VICTIM??
imgflip.com

Enclaves to the rescue!



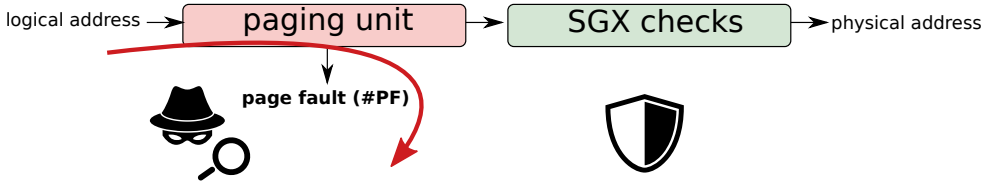
Intel SGX promise: hardware-level **isolation and attestation**

Intel SGX: A look under the hood



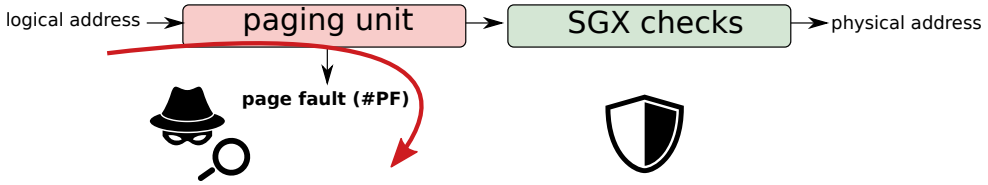
- **SGX machinery** protects against direct address remapping attacks

Intel SGX: A look under the hood



- **SGX machinery** protects against direct address remapping attacks
- . . . but untrusted address translation may **fault** during enclaved execution (!)

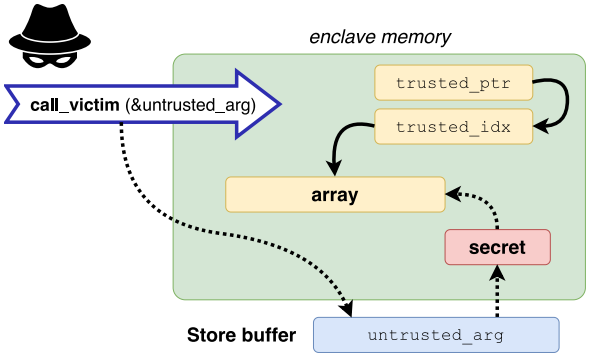
Intel SGX: A look under the hood



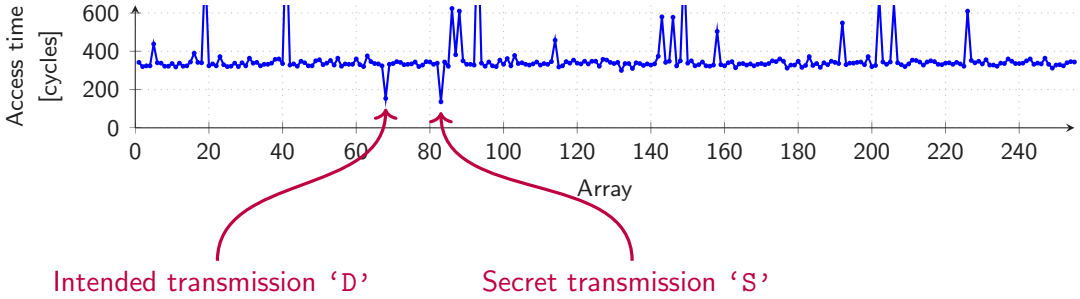
We can arbitrarily provoke page faults for trusted enclave loads!

LVI toy example: Hijacking transient data flow

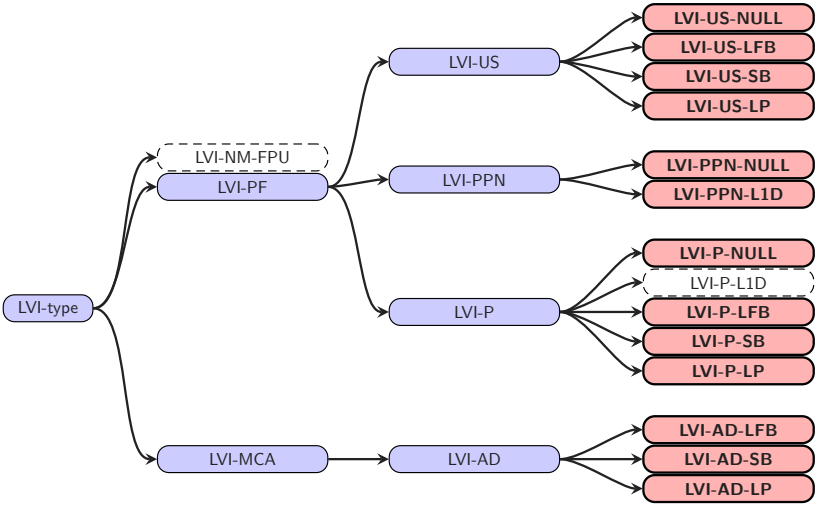
```
1 void call_victim(size_t
      untrusted_arg)
2 {
3   *arg_copy = untrusted_arg;
4   array[**trusted_ptr * 4096];
5 }
```



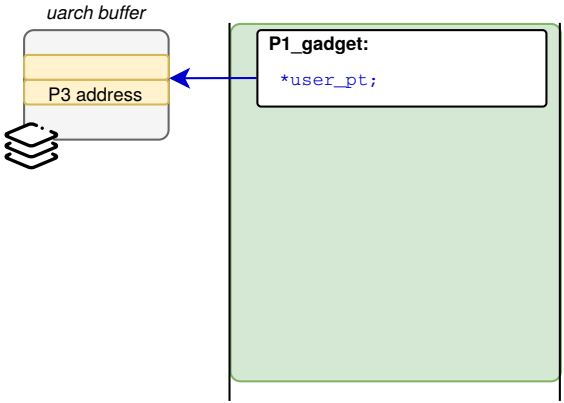
LVI toy example: Recovering arbitrary secrets



Taxonomy of LVI variants: Many buffers, many faults...

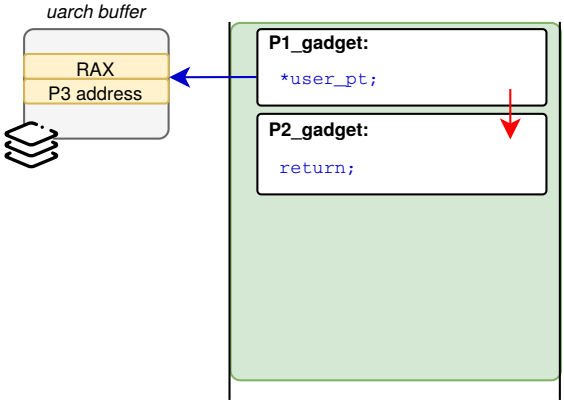


LVI-based transient control-flow hijacking



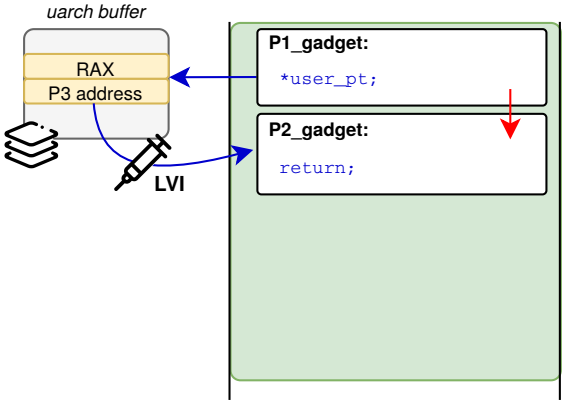
1. Victim fills μ-arch buffer with attacker-controlled data

LVI-based transient control-flow hijacking



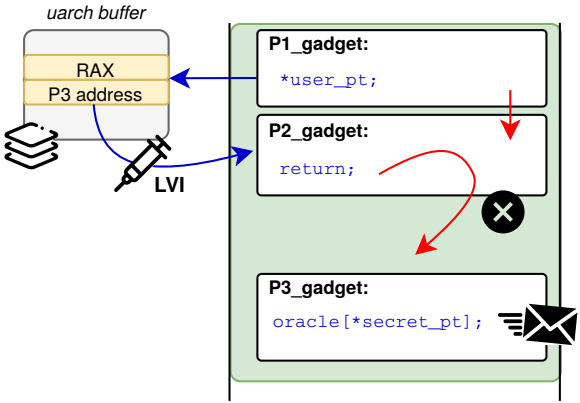
1. Victim fills μ-arch buffer with attacker-controlled data
2. Victim executes indirect branch (`JMP/CALL/RET`)

LVI-based transient control-flow hijacking

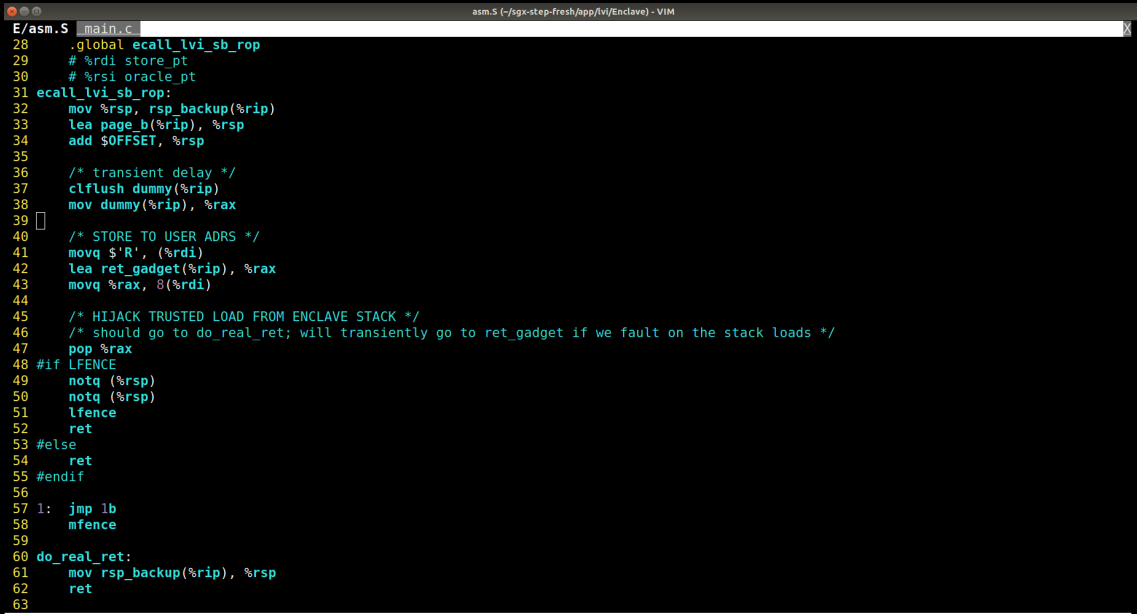


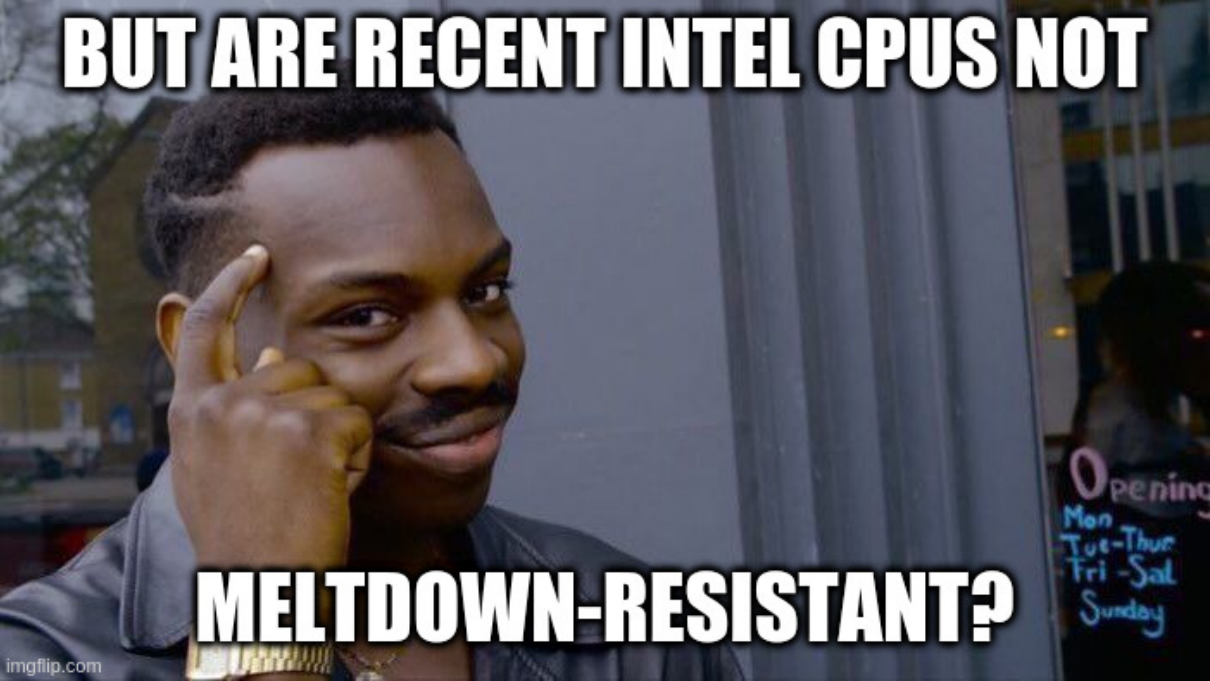
1. Victim fills μ-arch buffer with attacker-controlled data
2. Victim executes indirect branch (`JMP/CALL/RET`)
3. Faulting load $\rightarrow$ inject incorrect attacker values(!)

LVI-based transient control-flow hijacking

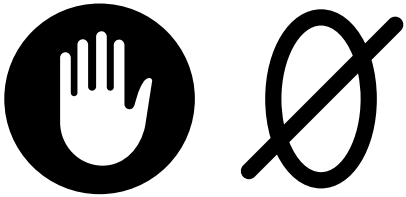


1. Victim fills μ-arch buffer with attacker-controlled data
2. Victim executes indirect branch (`JMP/CALL/RET`)
3. Faulting load $\rightarrow$ inject incorrect attacker values(!)
4. Redirect transient control flow


asm.S (~/sgx-step-fresh/app/lvi/Enclave) - VIM
E/asm.S
main.c
28 .global ecall_lvi_sb_rop
29 # %rdi store_pt
30 # %rsi oracle_pt
31 ecall_lvi_sb_rop:
32 mov %rsp, rsp_backup(%rip)
33 lea page_b(%rip), %rsp
34 add $OFFSET, %rsp
35
36 /* transient delay */
37 clflush dummy(%rip)
38 mov dummy(%rip), %rax
39
40 /* STORE TO USER ADRS */
41 movq $'R', (%rdi)
42 lea ret_gadget(%rip), %rax
43 movq %rax, 8(%rdi)
44
45 /* HIJACK TRUSTED LOAD FROM ENCLAVE STACK */
46 /* should go to do_real_ret; will transiently go to ret_gadget if we fault on the stack loads */
47 pop %rax
48 #if LFENCE
49 notq (%rsp)
50 notq (%rsp)
51 lfence
52 ret
53 #else
54 ret
55 #endif
56
57 1: jmp 1b
58 mfence
59
60 do_real_ret:
61 mov rsp_backup(%rip), %rsp
62 ret
63
Enclave/asm.S
39,0-1
84%

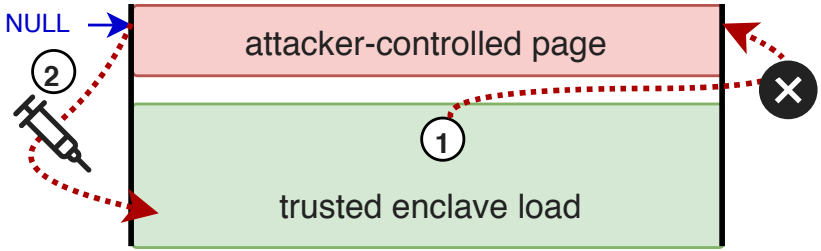

BUT ARE RECENT INTEL CPUS NOT
MELTDOWN-RESISTANT?
Opening
Mon
Tue-Thur
Fri -Sat
Sunday
imgflip.com

LVI-NULL: Why `0x00` is not a safe value



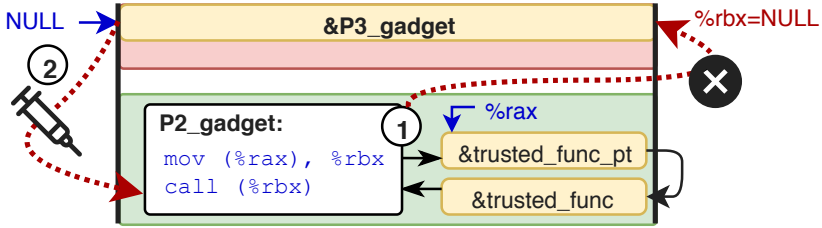
- Recent Intel CPUs forward `0x00` **dummy values** for faulting loads

LVI-NULL: Why 0x00 is not a safe value



- Recent Intel CPUs forward 0x00 **dummy values** for faulting loads
- . . . but NULL is a valid virtual memory address, under attacker control

LVI-NULL: Why 0x00 is not a safe value

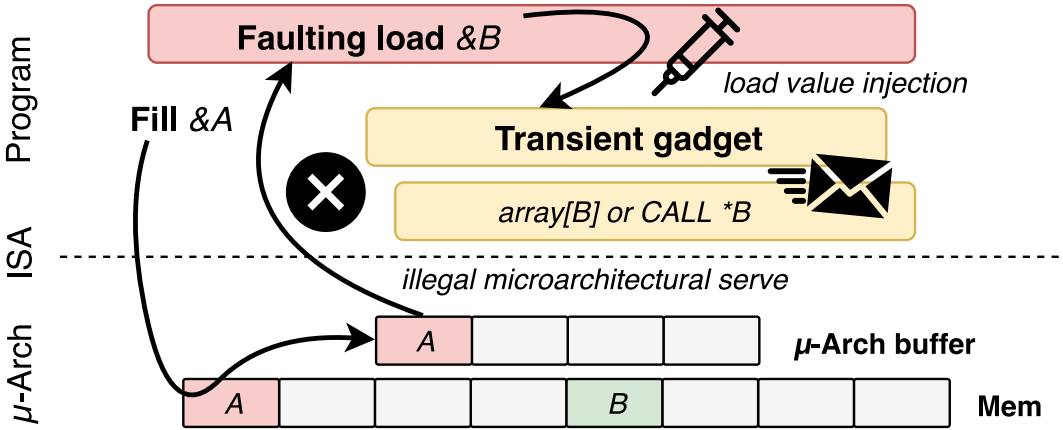


- Recent Intel CPUs forward 0x00 **dummy values** for faulting loads
- . . . but NULL is a valid virtual memory address, under attacker control
- . . . hijack pointer values (e.g., function pointer-to-pointer)

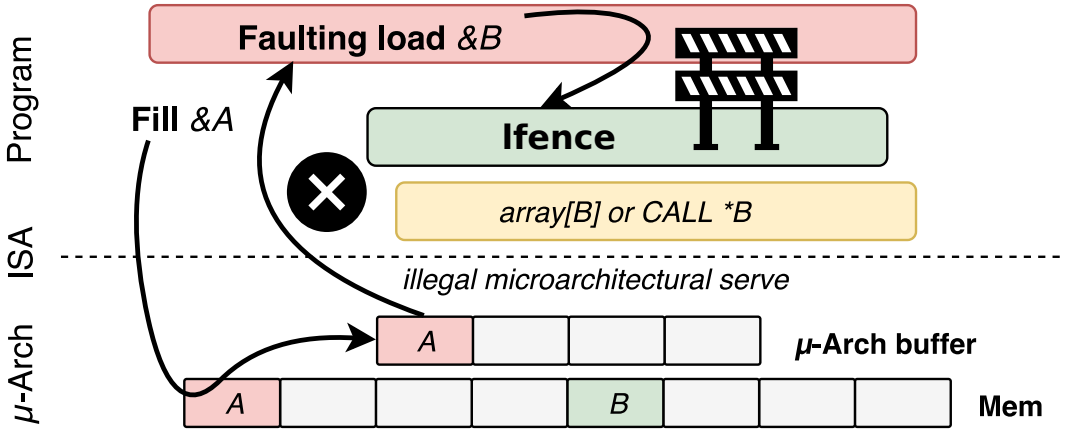
Mitigating LVI?

Gadget-based exploits $\rightarrow$ **flushing μ-arch buffers does not suffice!**

Mitigation idea: Fencing vulnerable load instructions



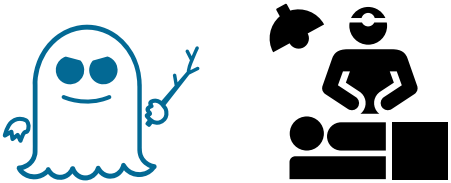
Mitigation idea: Fencing vulnerable load instructions



Serializing indirect branches

LVI ↔ Spectre: no control-flow prediction; **every load can be hijacked**

Instruction	Possible emulation	Clobber-free
`ret`	`pop %reg; lfence; jmp *%reg`	✗
`ret`	`not (%rsp); not (%rsp); lfence; ret`	✓
`jmp (mem)`	`mov (mem),%reg; lfence; jmp *%reg`	✗
`call (mem)`	`mov (mem),%reg; lfence; call *%reg`	✗

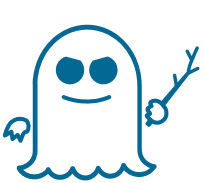


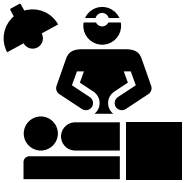
23 fences

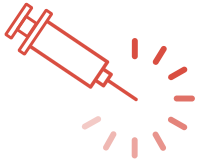
October 2019—"surgical precision"

Intel architectural enclaves: `lfence` counts

`libsgx_qe.signed.so`







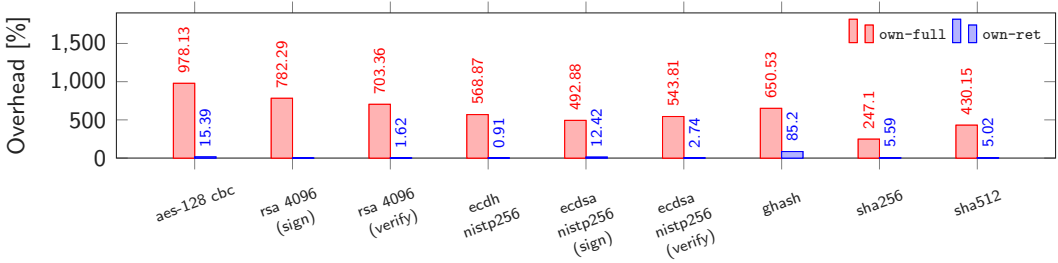
23 fences

October 2019—"surgical precision"

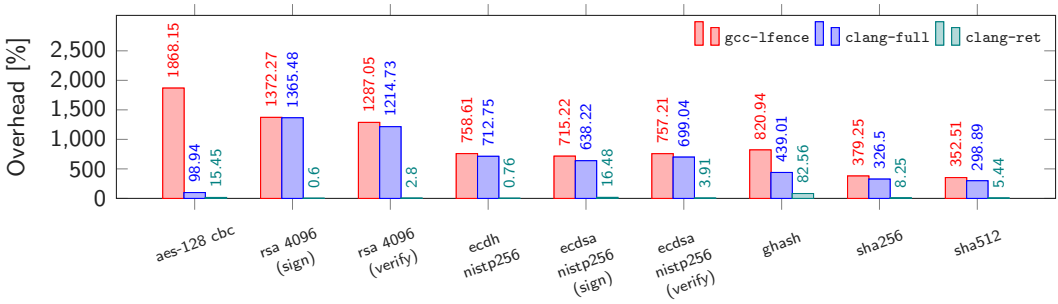
49,315 fences

March 2020—"big hammer"

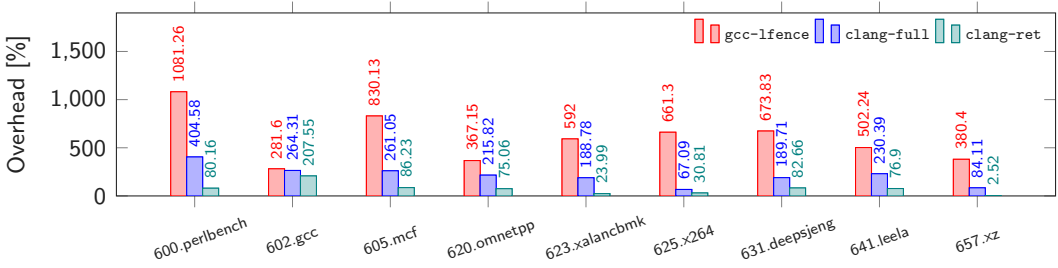
Performance overheads: OpenSSL (our prototype mitigation)



Performance overheads: OpenSSL (Intel's mitigation)



Performance overheads: SPEC (Intel's mitigation)



GNU Assembler Adds New Options For Mitigating Load Value Injection Attack

Written by Michael Larabel in GNU on 11 March 2020 at 02:55 PM EDT. 14 Comments

The Brutal Performance Impact From Mitigating The LVI Vulnerability

Written by Michael Larabel in Software on 12 March 2020. **Page 1 of 6**. 76 Comments

LLVM Lands Performance-Hitting Mitigation For Intel LVI Vulnerability

Written by Michael Larabel in Software on 3 April 2020. **Page 1 of 3**. 20 Comments

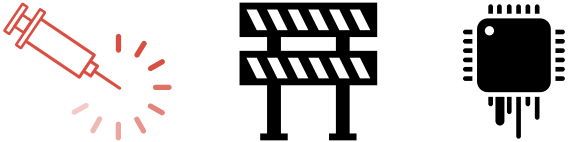
Looking At The LVI Mitigation Impact On Intel Cascade Lake Refresh

Written by Michael Larabel in Software on 5 April 2020. **Page 1 of 5**. 10 Comments

⇒ LVI **gadgets** reversely exploit Meltdown-type effects

⇒ **Short-term:** extensive `lfence` compiler mitigations for Intel SGX enclaves

⇒ **Long-term:** improved silicon patches in new CPUs



LVI

Hijacking Transient Execution through Microarchitectural Load Value Injection

Jo Van Bulck [1] **Daniel Moghimi** [2] **Michael Schwarz** [3] **Moritz Lipp** [3] **Marina Minkin** [4]
Daniel Genkin [4] **Yuval Yarom** [5] **Berk Sunar** [2] **Daniel Gruss** [3] **Frank Piessens** [1]

41st IEEE Symposium on Security and Privacy (digital edition) – May 18, 2020

[1]imec-DistriNet, KU Leuven [2]Worcester Polytechnic Institute [3]Graz University of Technology
[4]University of Michigan [5]University of Adelaide and Data61