



Microarchitectural Side-Channel Attacks for Privileged Software Adversaries

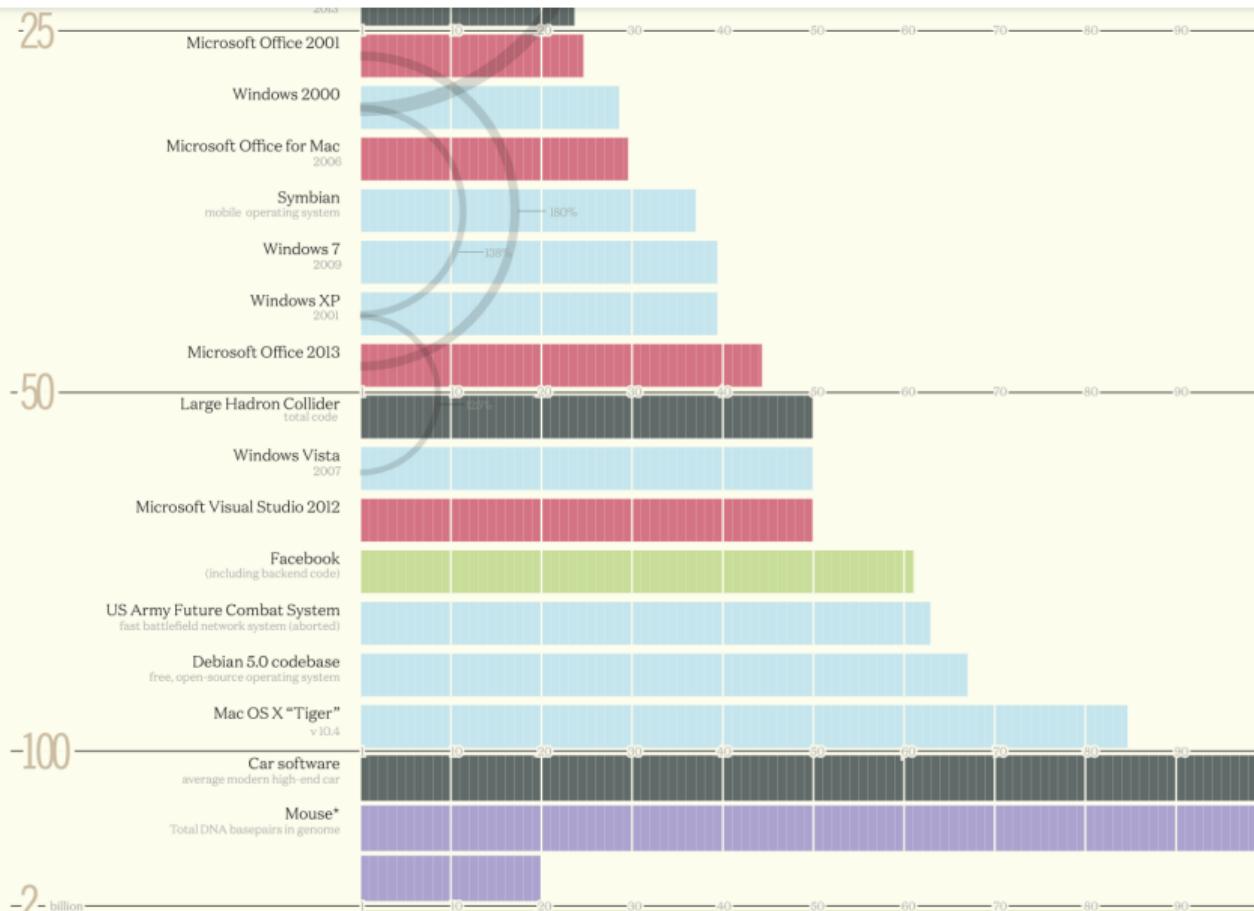
Jo Van Bulck

STM PhD Award Talk (online), October 8, 2021

🏡 imec-DistriNet, KU Leuven 📩 jo.vanbulck@cs.kuleuven.be 🐦 [jovanbulck](https://twitter.com/jovanbulck)

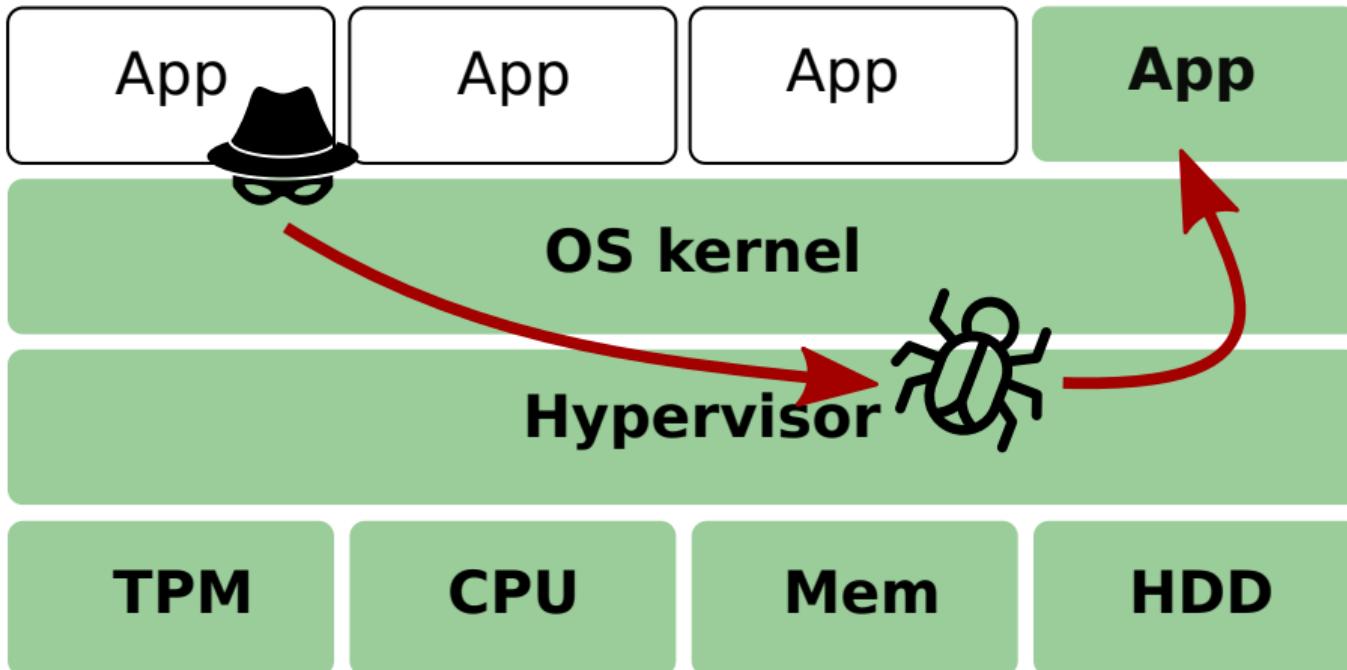
“Complexity is the worst enemy of security, and our systems are getting more complex all the time.”

— Bruce Schneier



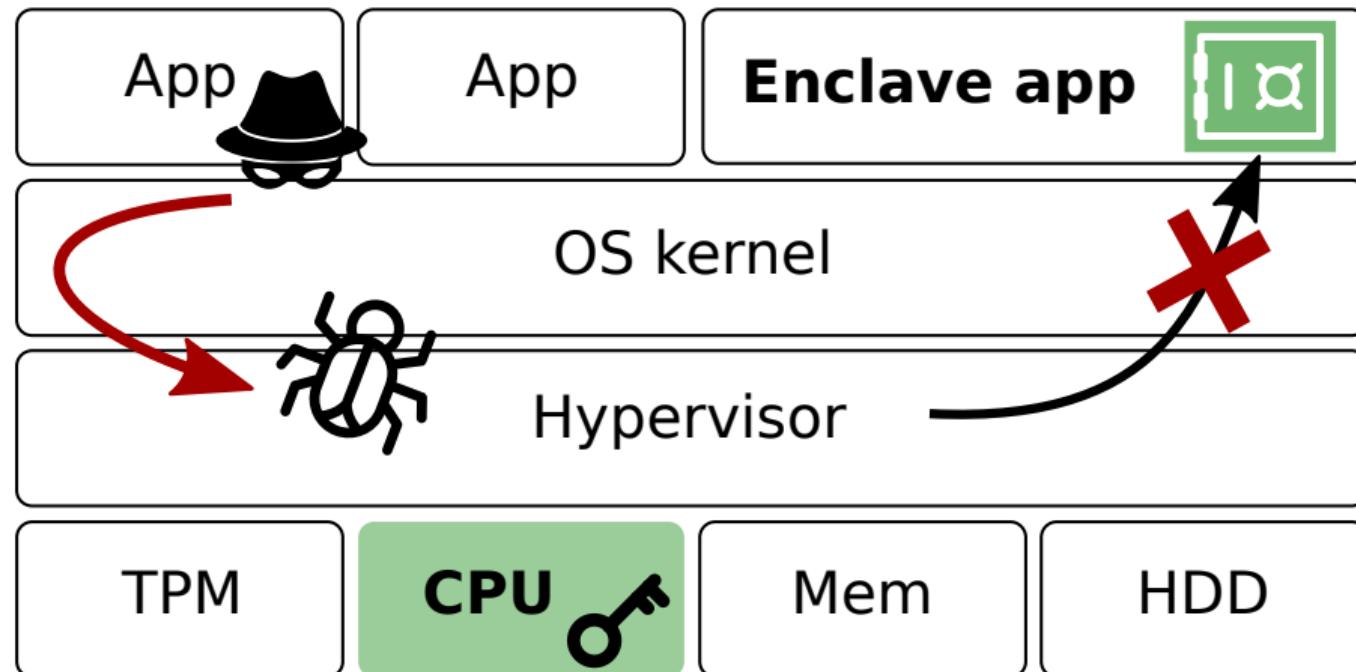


Enclaved execution: Reducing attack surface



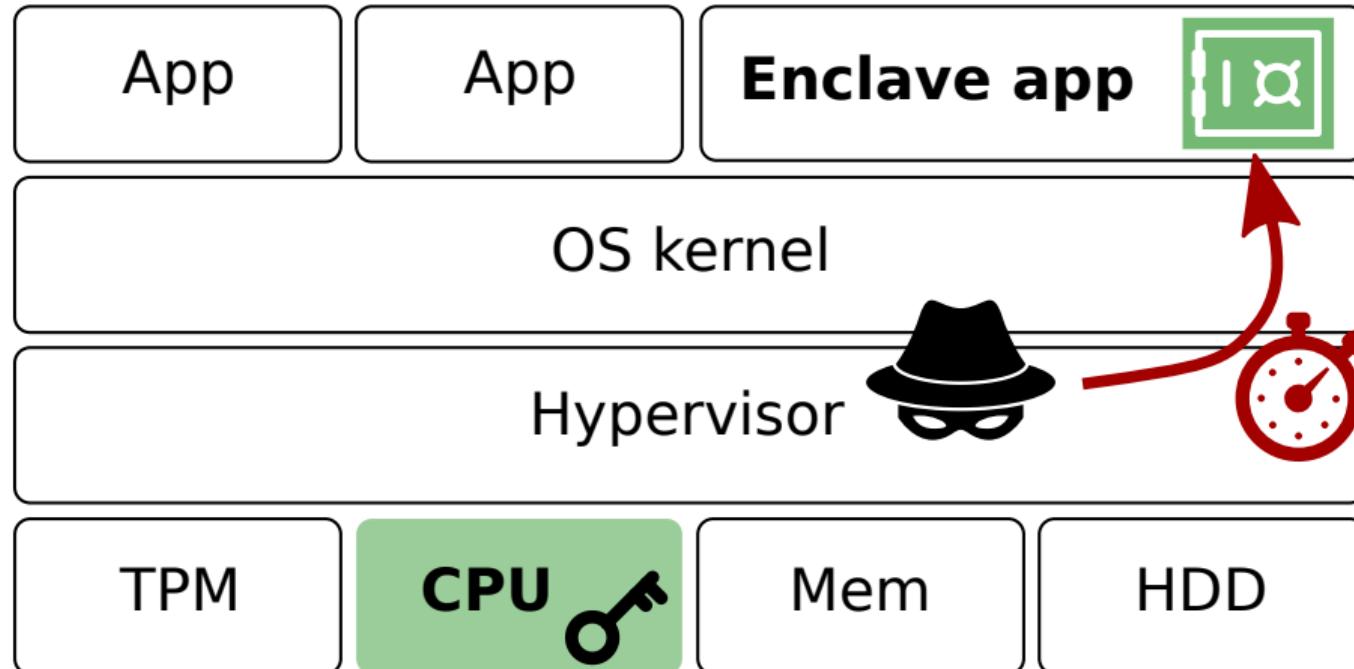
Traditional layered designs: large **trusted computing base**

Enclaved execution: Reducing attack surface



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

Enclaved execution: Privileged side-channel attacks



Game-changer: Untrusted OS → new class of powerful **side channels!**



Vault Door

WEIGHT: 22 1/2 Tons

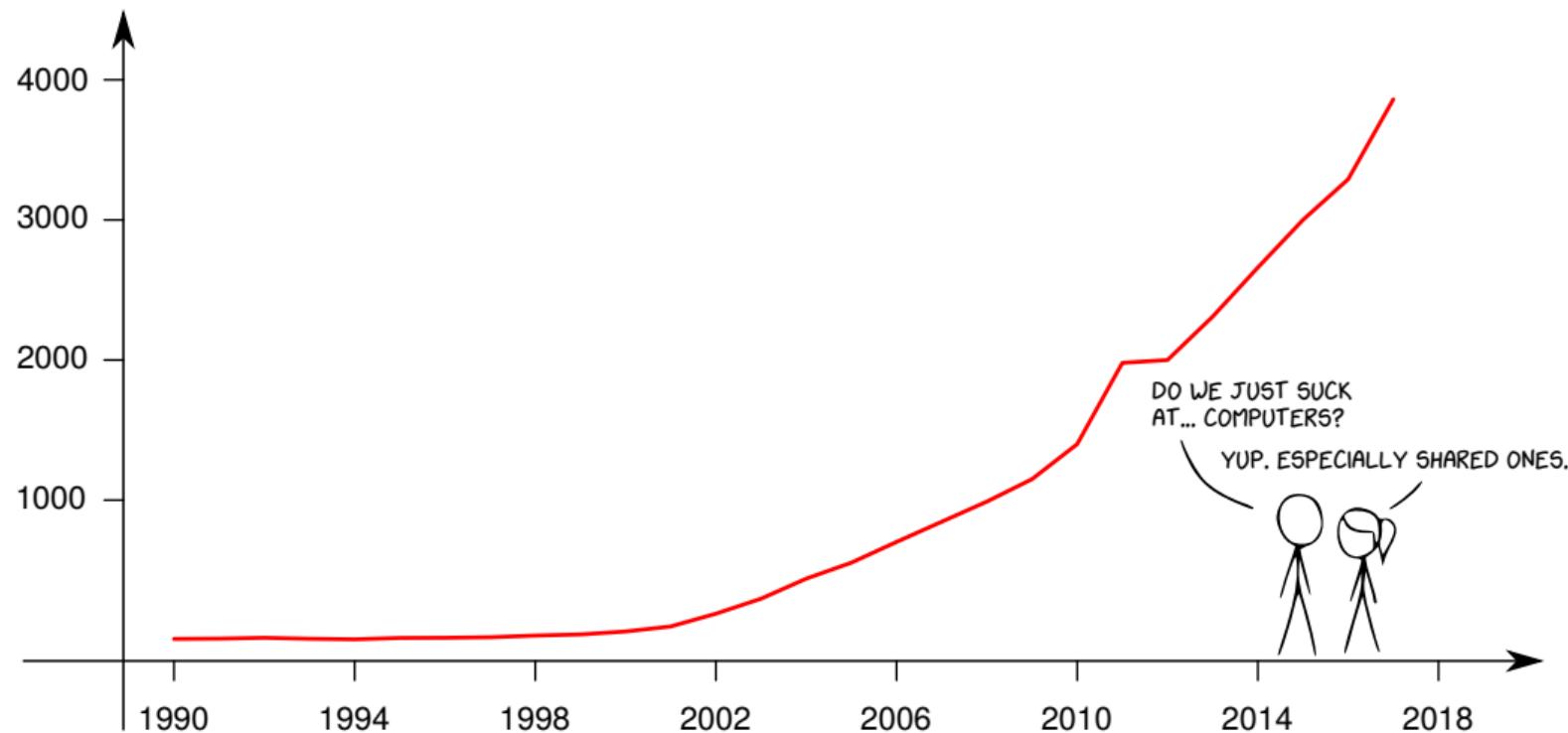
THICKNESS: 22 Inches

STEEL: 11 Layers of Special
Cutting and Drill Resistant

LOCKS: 4 Hamilton Watch
Movements for Time Locks

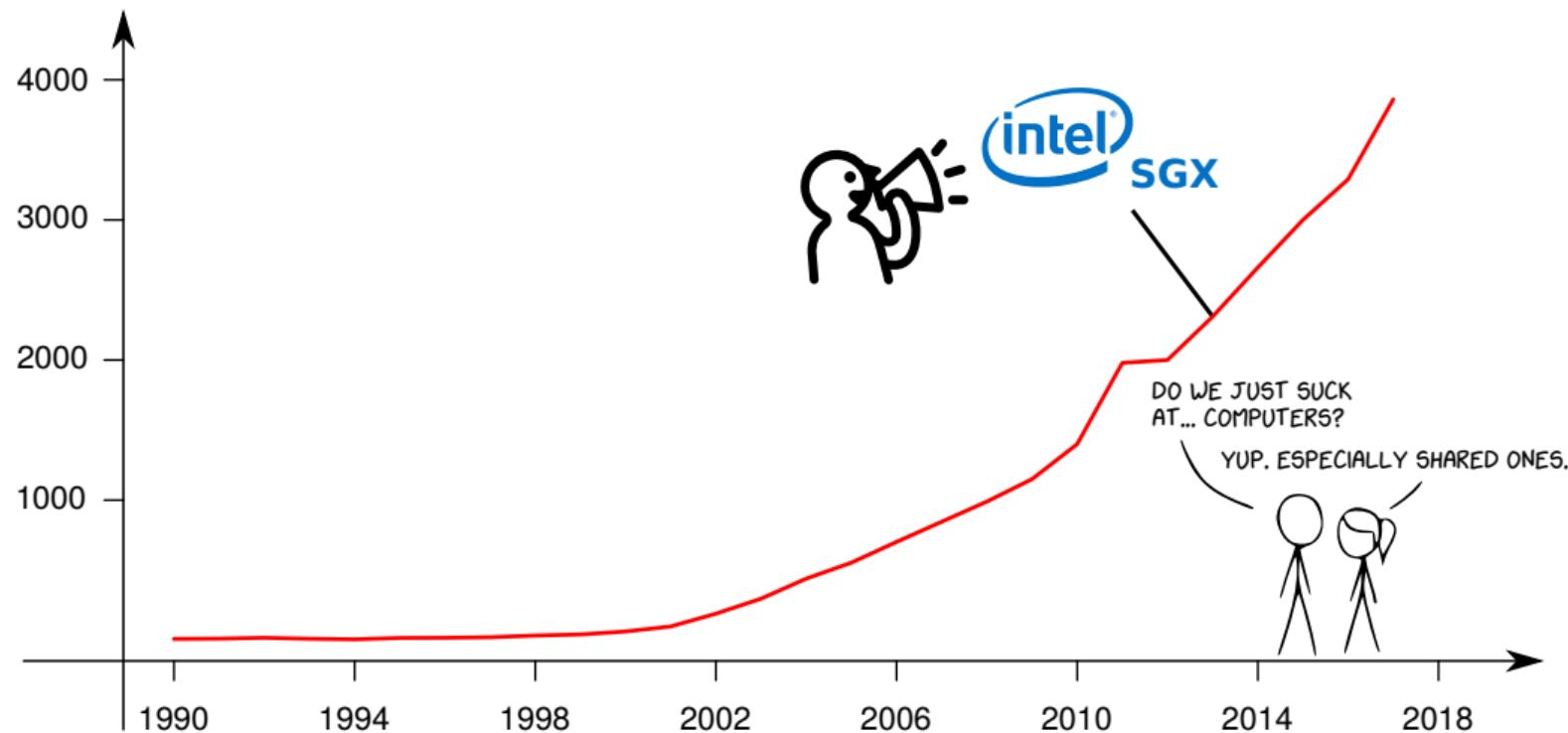


Evolution of “side-channel attack” research



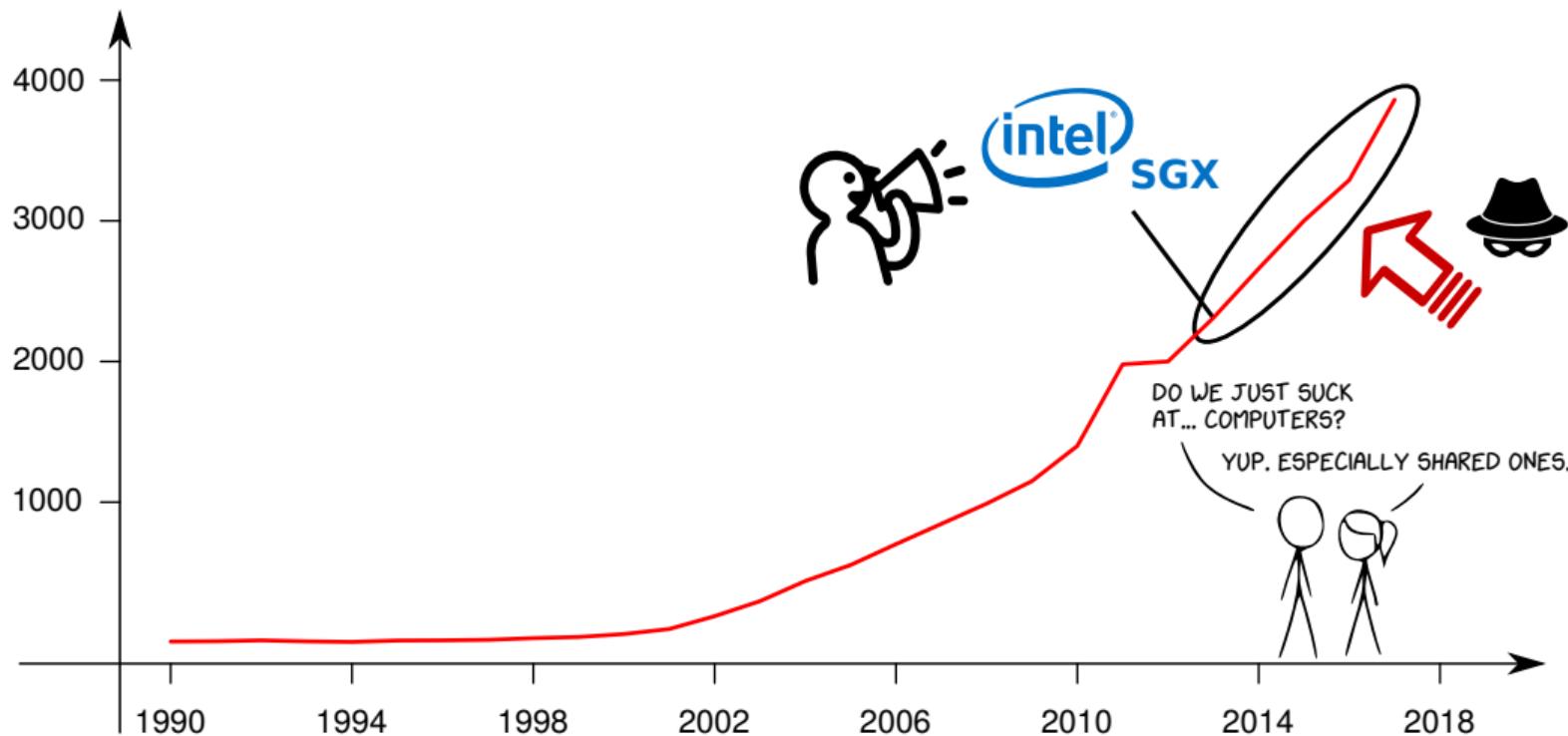
Based on github.com/Pold87/academic-keyword-occurrence and xkcd.com/1938/

Evolution of “side-channel attack” research



Based on github.com/Pold87/academic-keyword-occurrence and xkcd.com/1938/

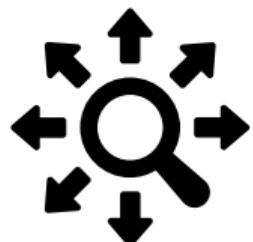
Side-channel attacks and trusted computing (focus of this PhD)



Based on github.com/Pold87/academic-keyword-occurrence and xkcd.com/1938/

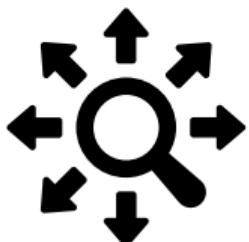


Research agenda: Understanding privileged side-channel attacks



1. **Which** novel privileged side channels exist?
2. **How** well can they be exploited in practice?
3. **What** can be leaked?

Research agenda: Understanding privileged side-channel attacks



1. **Which** novel privileged side channels exist?
 - We uncover previously **unknown attack avenues**
2. **How** well can they be exploited in practice?
 - We develop **new techniques** and practical attack frameworks
3. **What** can be leaked?
 - We leak **metadata** and **data**



Idea 1: Privileged interrupts for side-channel amplification

Case study: Comparing a secret password



password

The word "password" is composed of seven light green rounded rectangular boxes, each containing one letter. The letters are black and arranged horizontally from left to right.

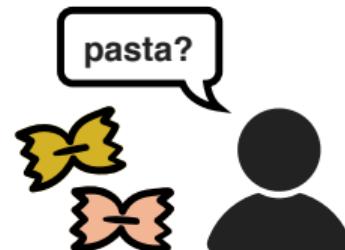
Case study: Comparing a secret password

password

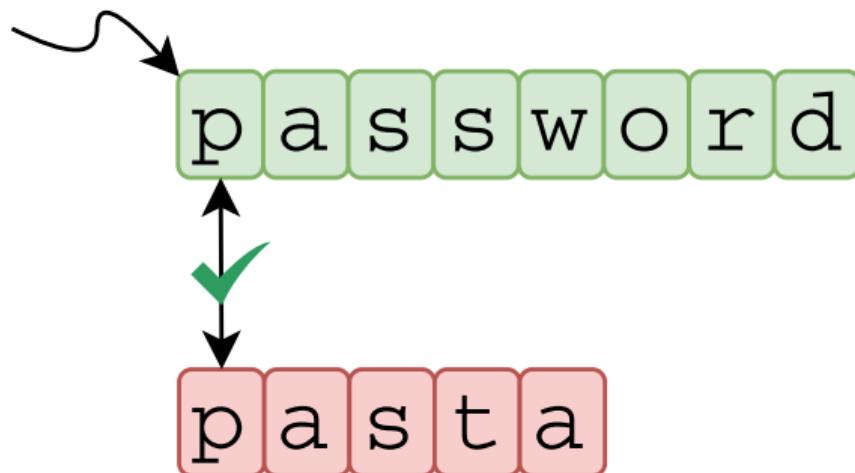


No!

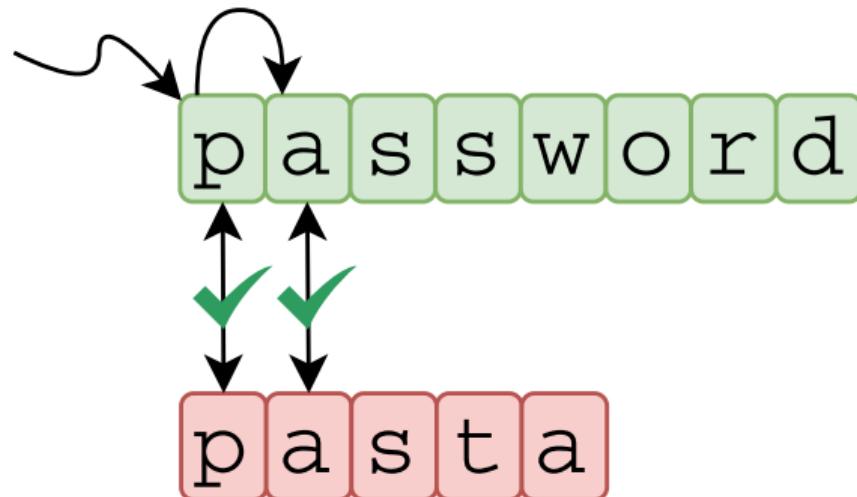
pasta



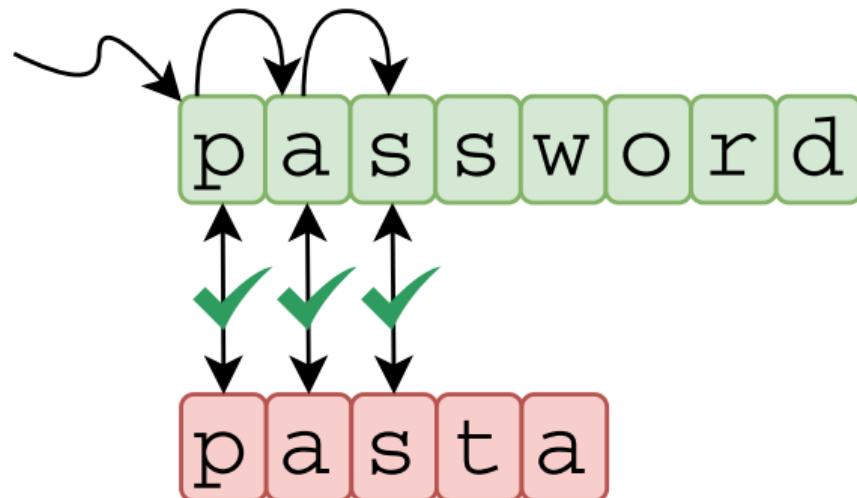
Case study: Comparing a secret password



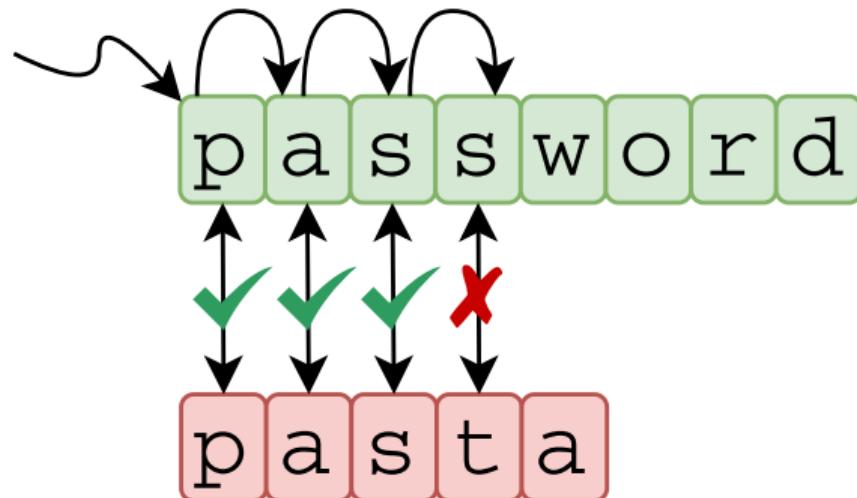
Case study: Comparing a secret password



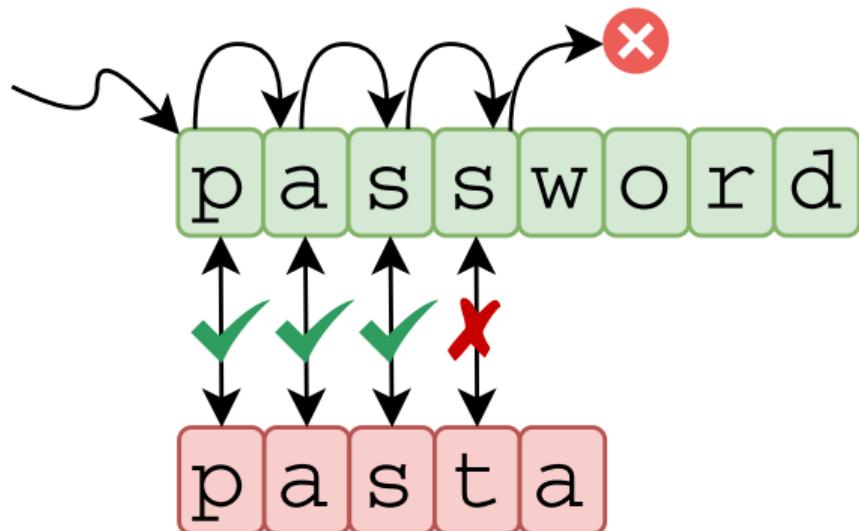
Case study: Comparing a secret password



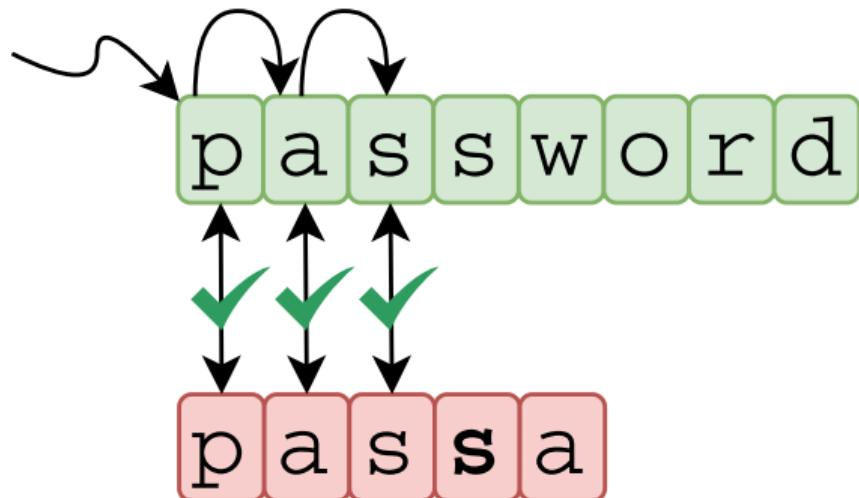
Case study: Comparing a secret password



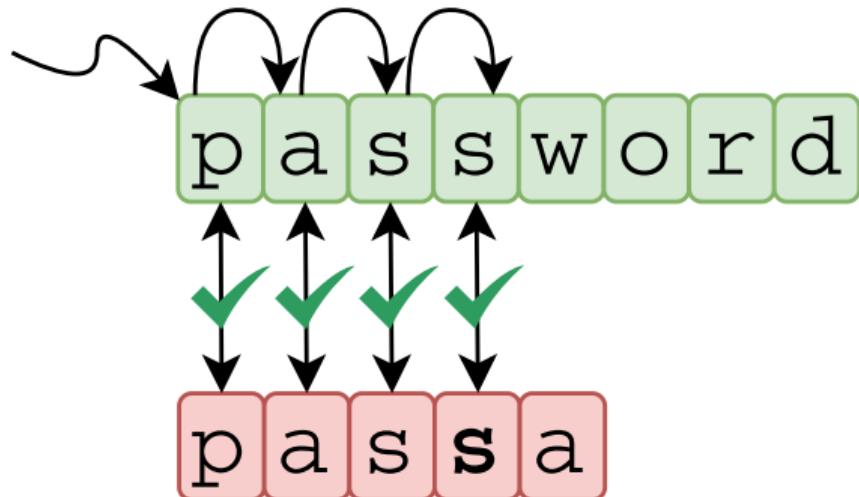
Case study: Comparing a secret password



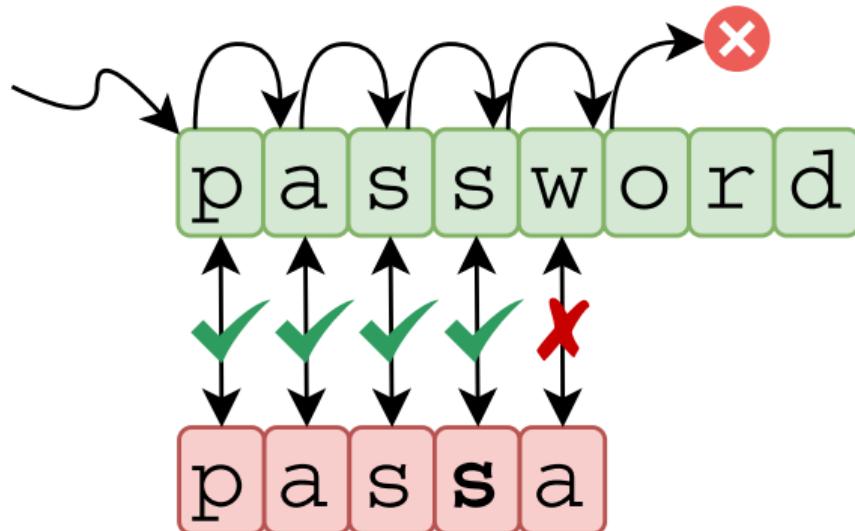
Case study: Comparing a secret password



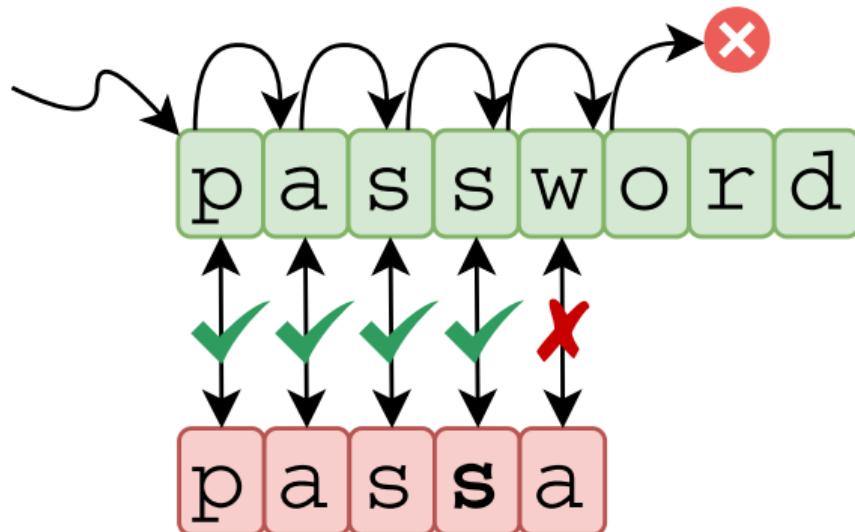
Case study: Comparing a secret password



Case study: Comparing a secret password

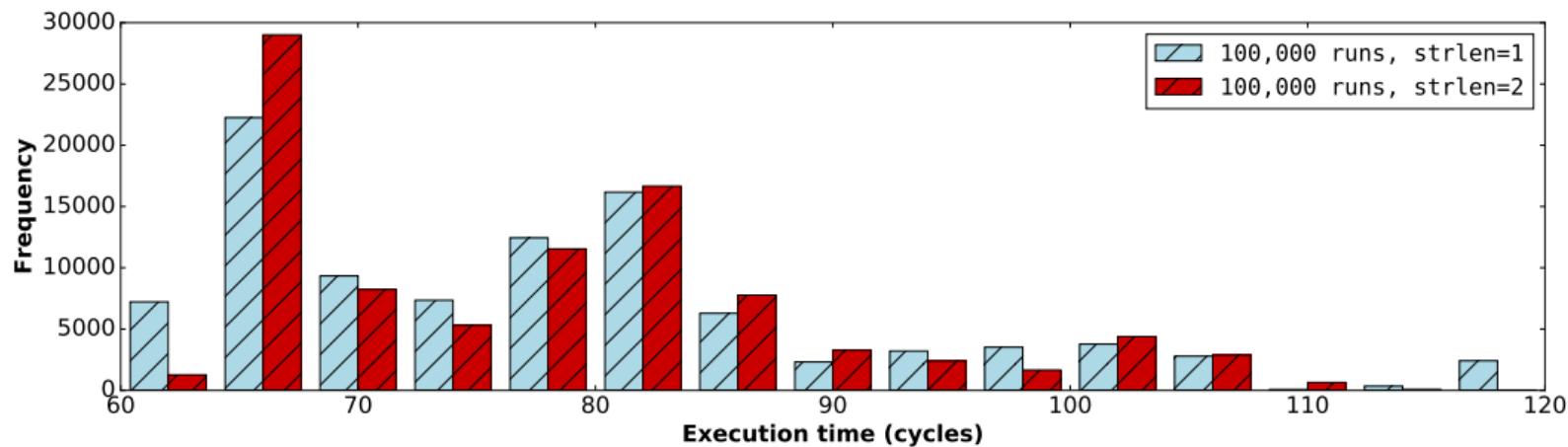


Case study: Comparing a secret password



Overall **execution time** reveals correctness of individual password bytes!

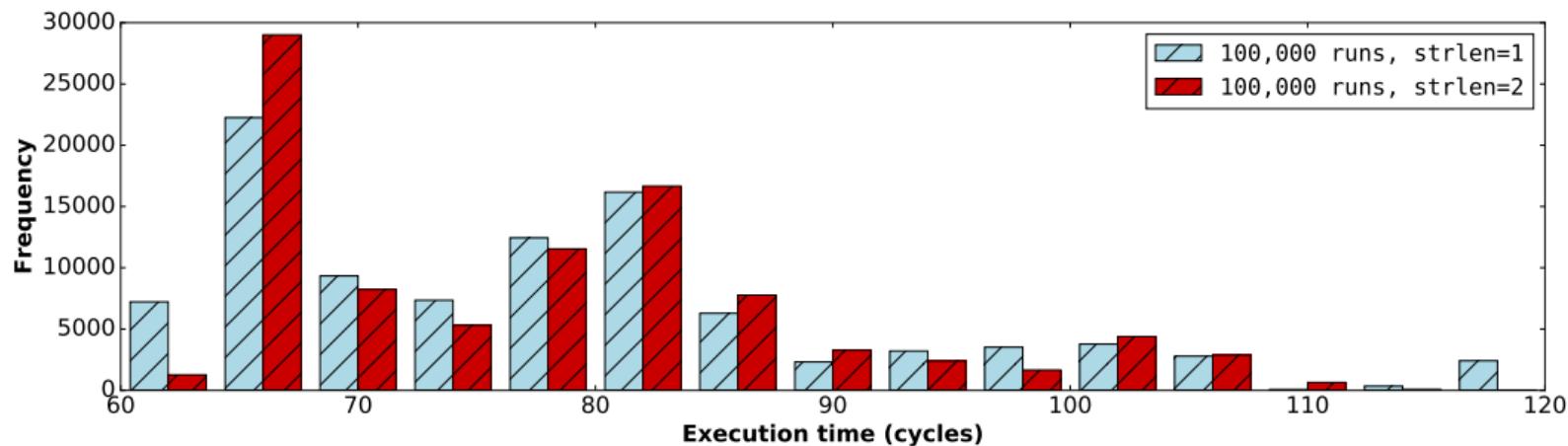
Building the side-channel oracle with execution timing?



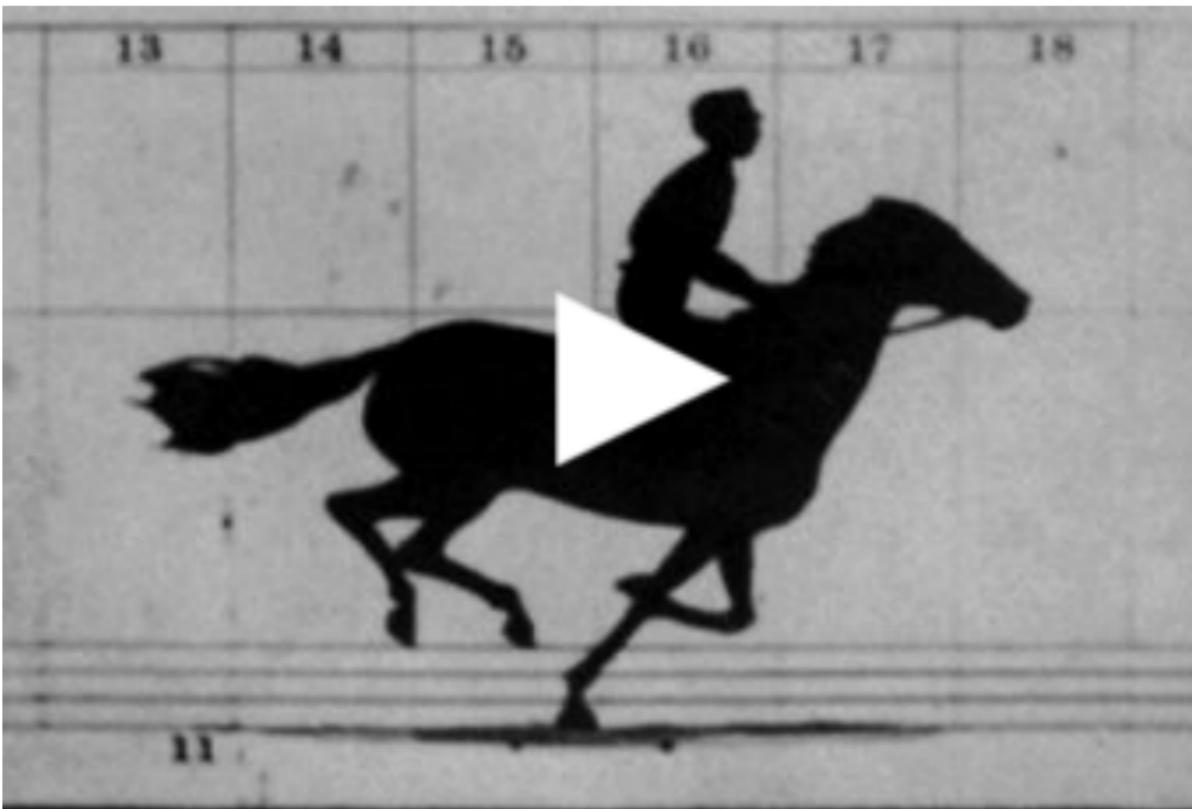
Building the side-channel oracle with execution timing?



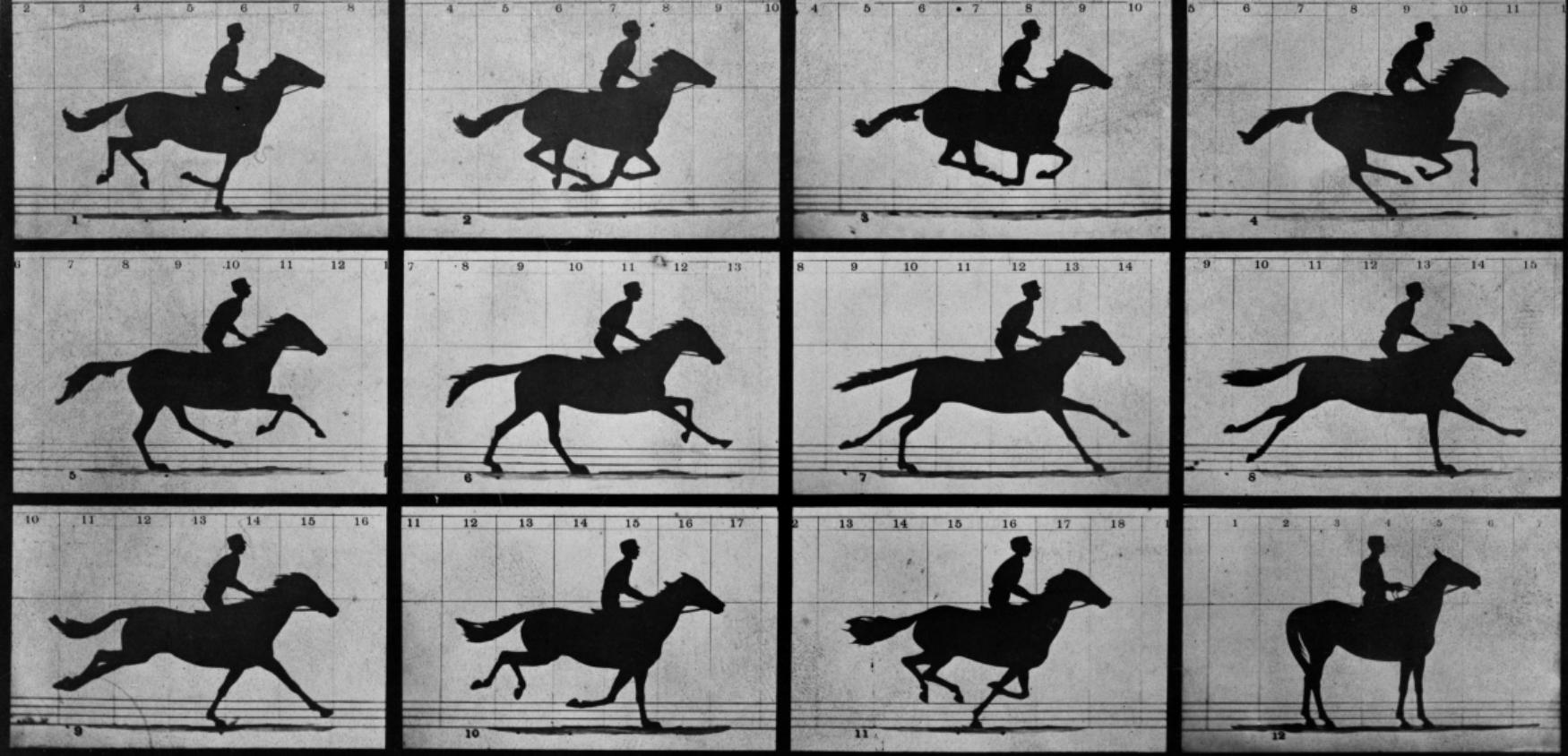
Too noisy: modern x86 processors are lightning fast...



Analogy: Studying galloping horse dynamics



https://en.wikipedia.org/wiki/Sallie_Gardner_at_a_Gallop



Copyright, 1878, by MUYBRIDGE.

MORSE'S Gallery, 417 Montgomery St., San Francisco.

THE HORSE IN MOTION.

Illustrated by
MUYBRIDGE

"SALLIE GARDNER," owned by LELAND STANFORD; running at a 1.40 gait over the Palo Alto track, 19th June, 1878.

The sequence of the photographs was taken at the rate of 12 exposures per second, or 120 exposures per second, so that, in one second of time, there illustrate successive positions of the horse in motion.

SGX-Step: Executing enclaves one instruction at a time



SGX-Step



<https://github.com/jovanbulck/sgx-step>



Unwatch ▾

27



Star

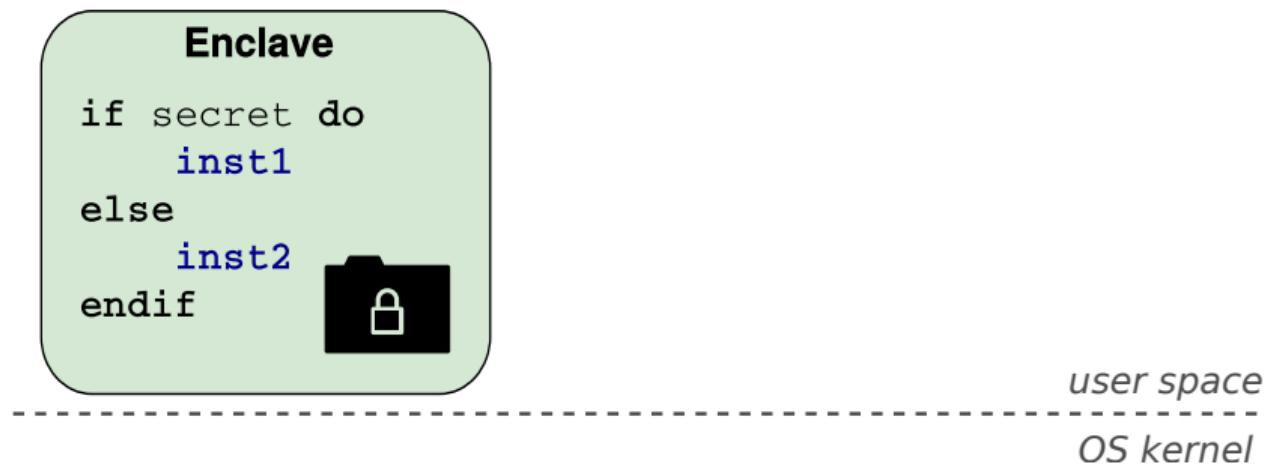
312



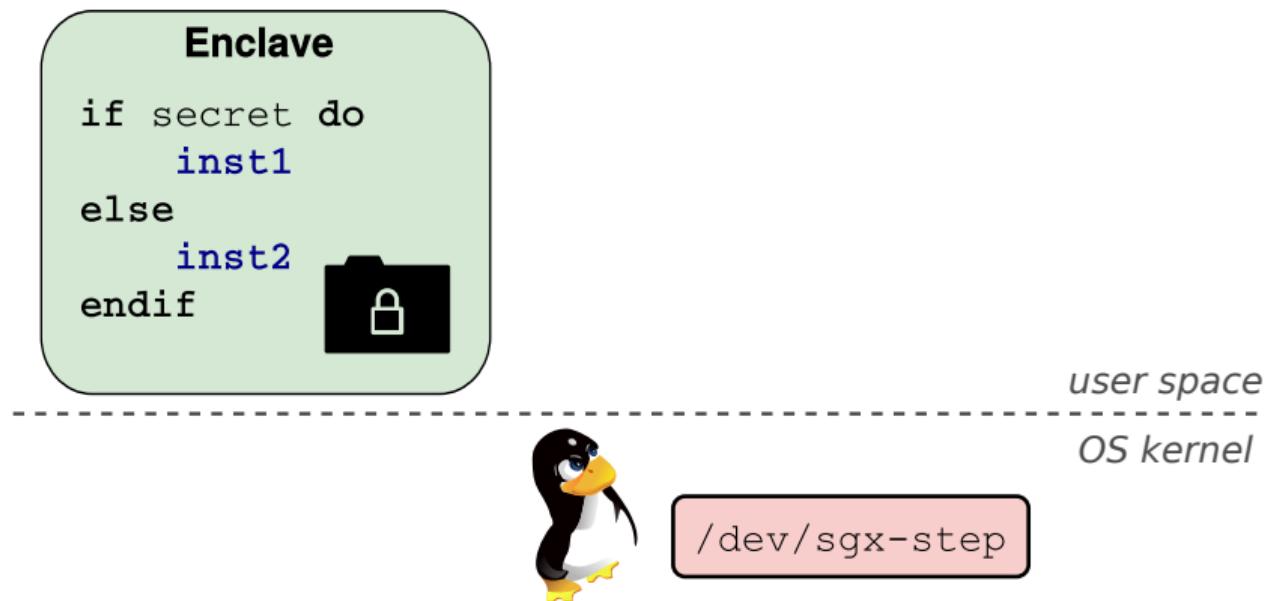
Fork

63

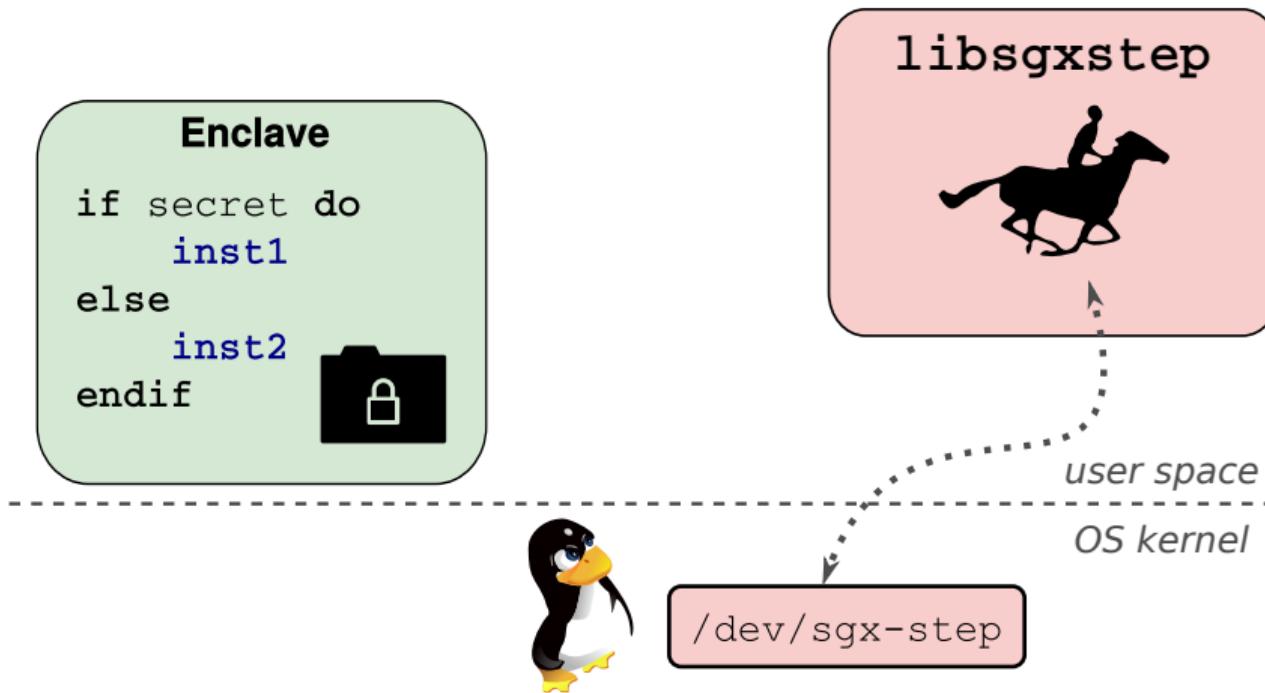
SGX-Step: Executing enclaves one instruction at a time



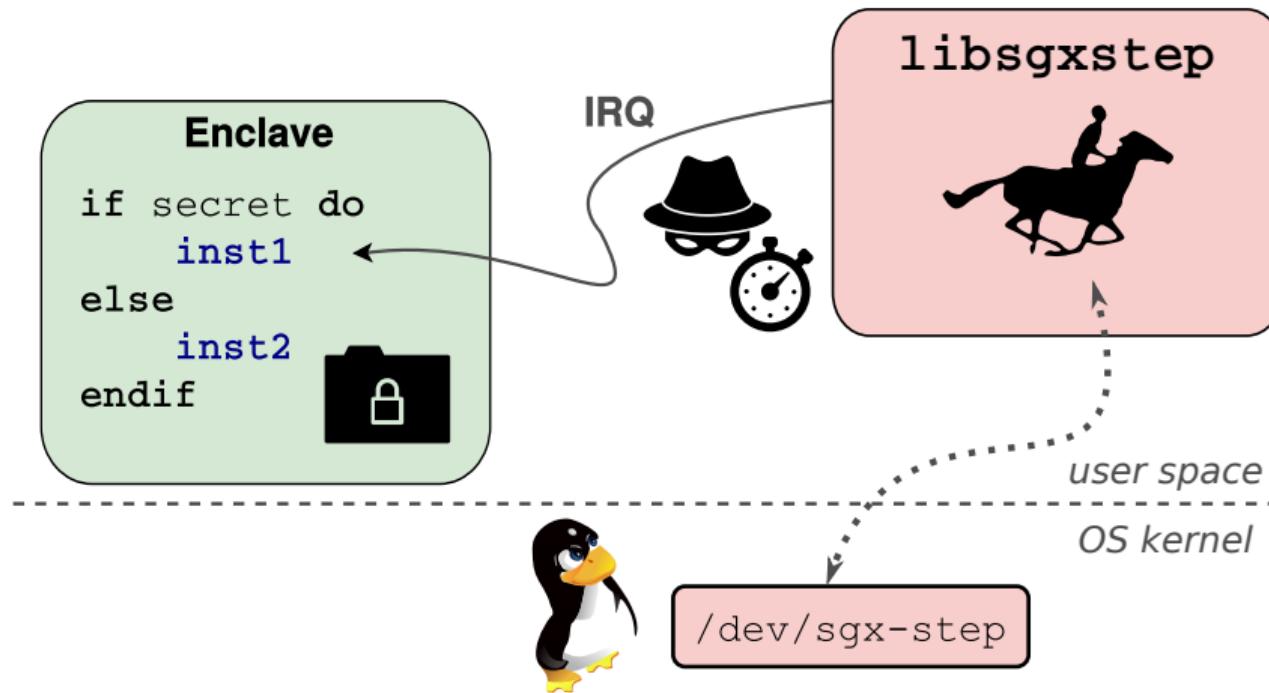
SGX-Step: Executing enclaves one instruction at a time



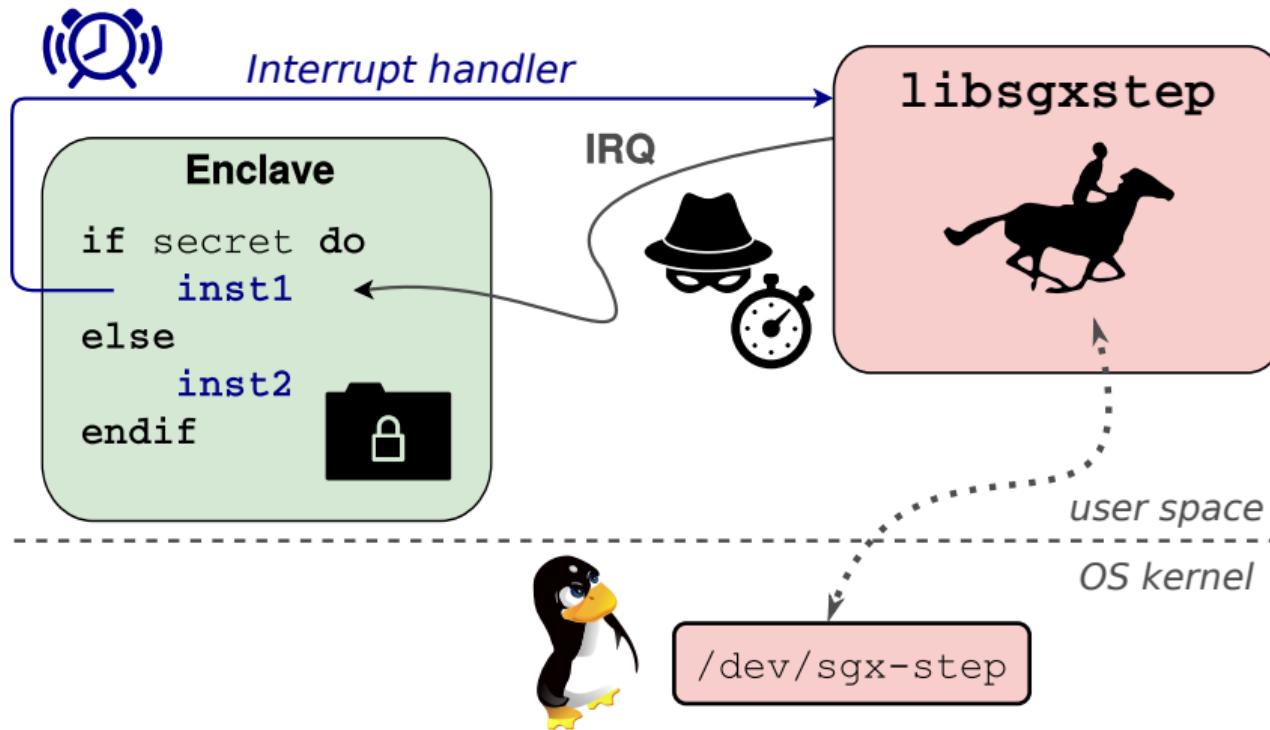
SGX-Step: Executing enclaves one instruction at a time



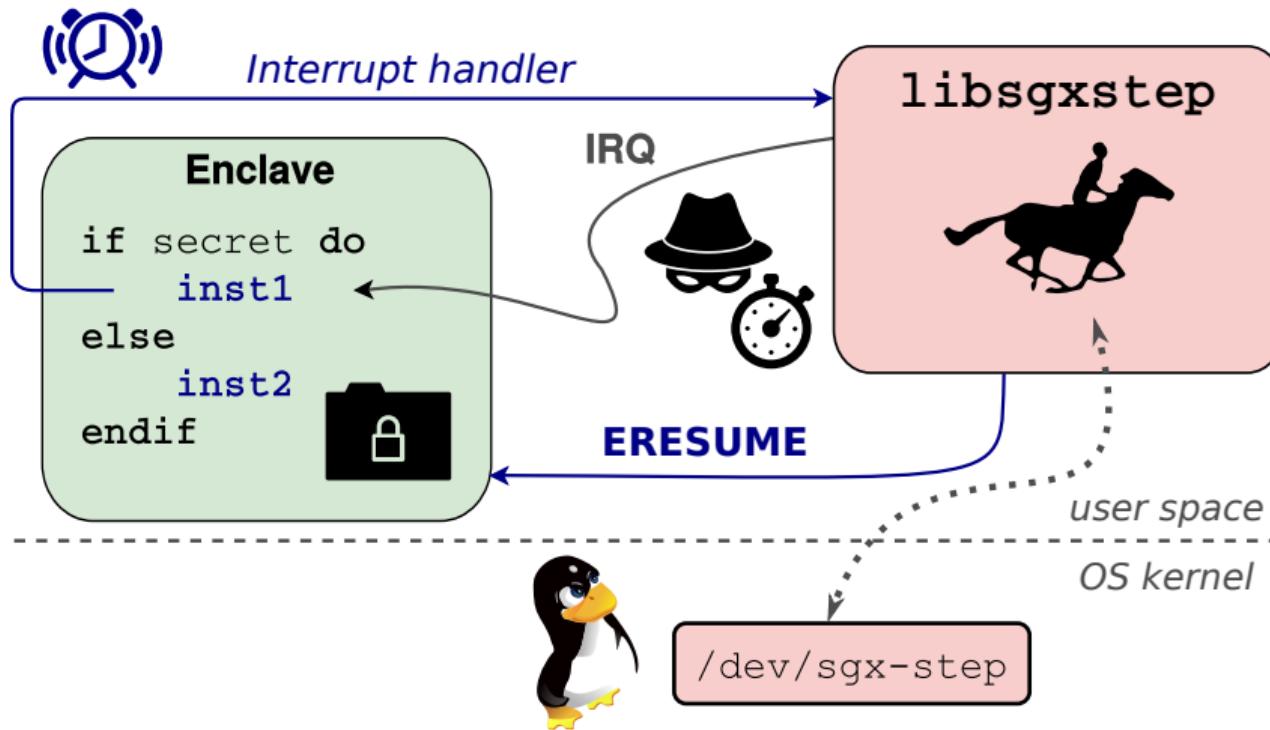
SGX-Step: Executing enclaves one instruction at a time



SGX-Step: Executing enclaves one instruction at a time



SGX-Step: Executing enclaves one instruction at a time



Demo: Building a deterministic password oracle with SGX-Step

```
[idt.c] DTR.base=0xfffffe0000000000/size=4095 (256 entries)
[idt.c] established user space IDT mapping at 0x7f7ff8e9a000
[idt.c] installed asm IRQ handler at 10:0x56312d19b000
[idt.c] IDT[ 45] @0x7f7ff8e9a2d0 = 0x56312d19b000 (seg sel 0x10); p=1; dpl=3; type=14; ist=0
[file.c] reading buffer from '/dev/cpu/1/msr' (size=8)
[apic.c] established local memory mapping for APIC_BASE=0xfee00000 at 0x7f7ff8e99000
[apic.c] APIC_ID=2000000; LVTT=400ec; TDCR=0
[apic.c] APIC timer one-shot mode with division 2 (lvtt=2d/tocr=0)
```

```
-----  
[main.c] recovering password length  
-----
```

```
[attacker] steps=15; guess='*****'  
[attacker] found pwd len = 6
```

```
-----  
[main.c] recovering password bytes  
-----
```

```
[attacker] steps=35; guess='SECRET' --> SUCCESS
```

```
[apic.c] Restored APIC_LVTT=400ec/TDCR=0
[file.c] writing buffer to '/dev/cpu/1/msr' (size=8)
[main.c] all done; counted 2260/2183 IRQs (AEP/IDT)
jo@breuer:~/sgx-step-demo$ █
```

ALL YOUR PASSWORDS



ARE BELONG TO US

SGX-Step: Enabling a new line of high-precision enclave attacks

Yr	Attack	Temporal resolution	APIC		PTE		Desc		Drv
			IRQ	IPI	#PF	A/D	PPN	GDT	
'15	Ctrl channel	~ Page	○	○	●	○	○	○	● ✓
'16	AsyncShock	~ Page	○	○	●	○	○	○	- ⚡
'17	CacheZoom	X > 1	●	○	○	○	○	○	✓ ⚡
'17	Hahnel et al.	X 0 - > 1	●	○	○	○	○	●	✓
'17	BranchShadow	X 5 - 50	●	○	○	○	○	○	✗ ⚡
'17	Stealthy PTE	~ Page	○	●	○	●	○	●	✓ ⚡
'17	DarkROP	~ Page	○	○	●	○	○	○	✓ ⚡
'17	SGX-Step	✓ 0 - 1	●	○	●	●	○	○	✓ ⚡
'18	Off-limits	✓ 0 - 1	●	○	●	○	○	●	✓ ⚡
'18	Single-trace RSA	~ Page	○	○	●	○	○	○	✓ ⚡
'18	Foreshadow	✓ 0 - 1	●	○	●	○	●	○	✓ ⚡
'18	SgxPectre	~ Page	○	○	●	○	○	○	✓ ⚡
'18	CacheQuote	X > 1	●	○	○	○	○	○	✓ ⚡
'18	SGXlinger	X > 1	●	○	○	○	○	○	✗ ⚡
'18	Nemesis	✓ 1	●	○	●	●	○	●	✓ ⚡

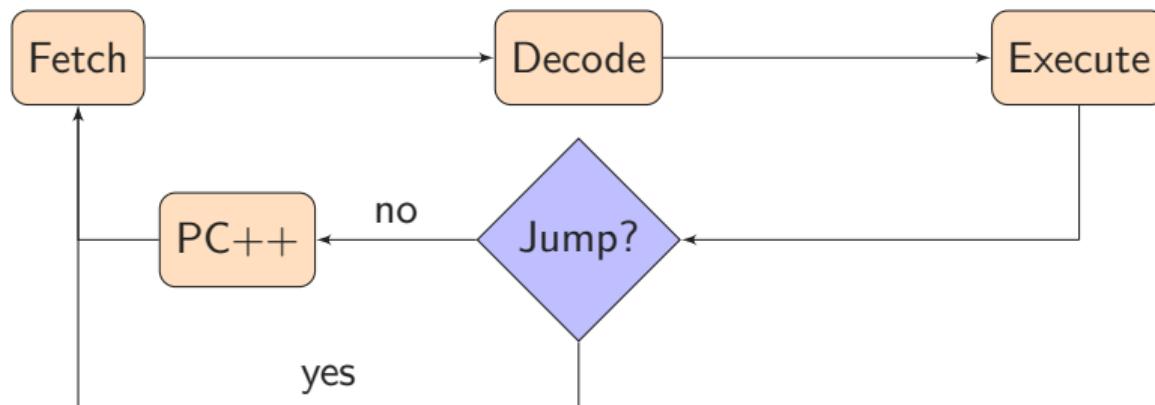
Yr	Attack	Temporal resolution	APIC		PTE		Desc		Drv
			IRQ	IPI	#PF	A/D	PPN	GDT	
'19	Spoiler	✓ 1	●	○	○	●	○	○	● ✓ ⚡
'19	ZombieLoad	✓ 0 - 1	●	○	●	●	○	○	● ✓ ⚡
'19	Tale of 2 worlds	✓ 1	●	○	●	●	○	○	● ✓ ⚡
'19	MicroScope	~ 0 - Page	○	○	●	○	○	○	✗ ⚡
'20	Bluethunder	✓ 1	●	○	○	○	○	○	● ✓ ⚡
'20	Big troubles	~ Page	○	○	●	○	○	○	○ ✓ ⚡
'20	Viral primitive	✓ 1	●	○	●	●	○	○	● ✓ ⚡
'20	CopyCat	✓ 1	●	○	●	●	○	○	● ✓ ⚡
'20	LVI	✓ 1	●	○	●	●	●	○	● ✓ ⚡
'20	A to Z	~ Page	○	○	●	○	○	○	○ ✓ ⚡
'20	Frontal	✓ 1	●	○	●	●	○	○	● ✓ ⚡
'20	CrossTalk	✓ 1	●	○	●	○	○	○	● ✓ ⚡
'20	Online template	~ Page	○	○	●	○	○	○	○ ✓ ⚡
'20	Déjà Vu NSS	~ Page	○	○	●	○	○	○	○ ✓ ⚡



Idea 2: Privileged interrupts for microarchitectural leakage

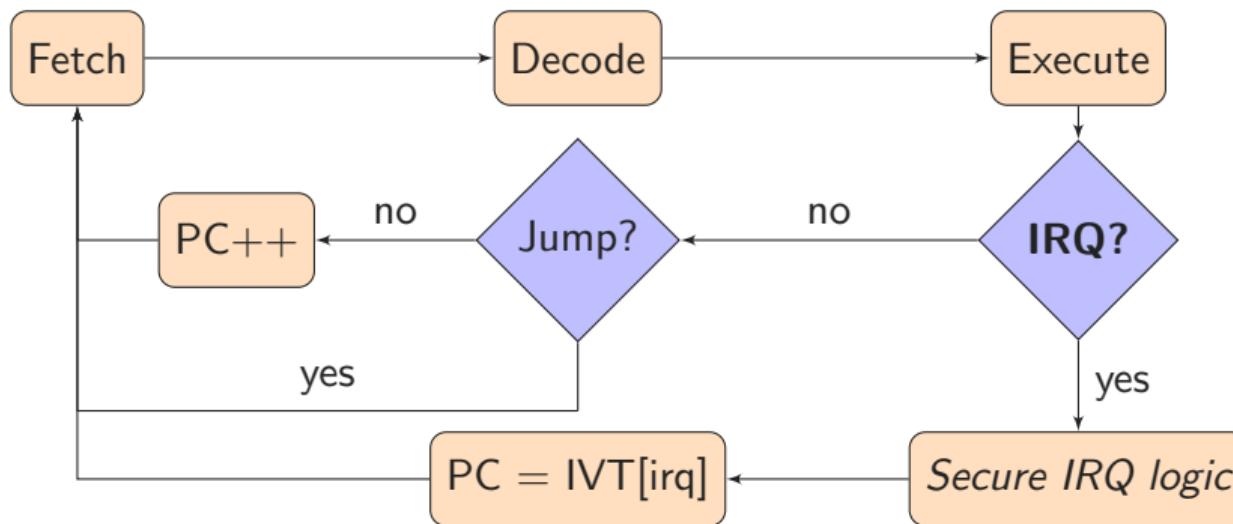
Back to basics: Fetch-decode-execute

Elementary CPU behavior: Stored program computer



Back to basics: Fetch-decode-execute

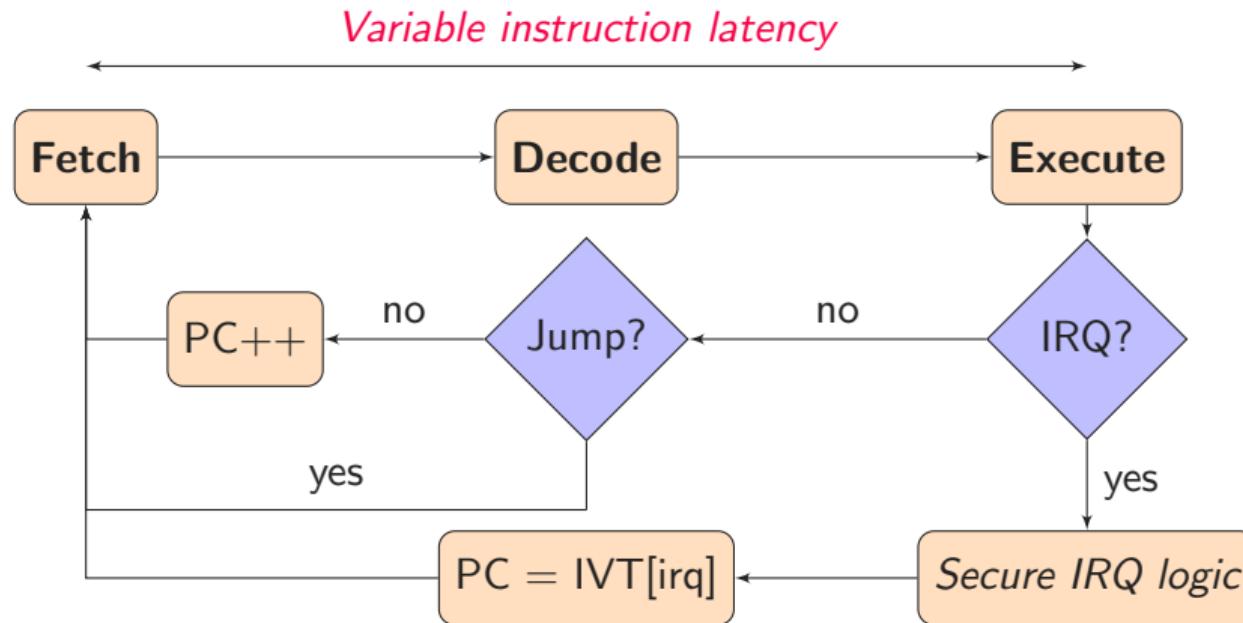
Interrupts: Asynchronous events, handled on instruction retirement



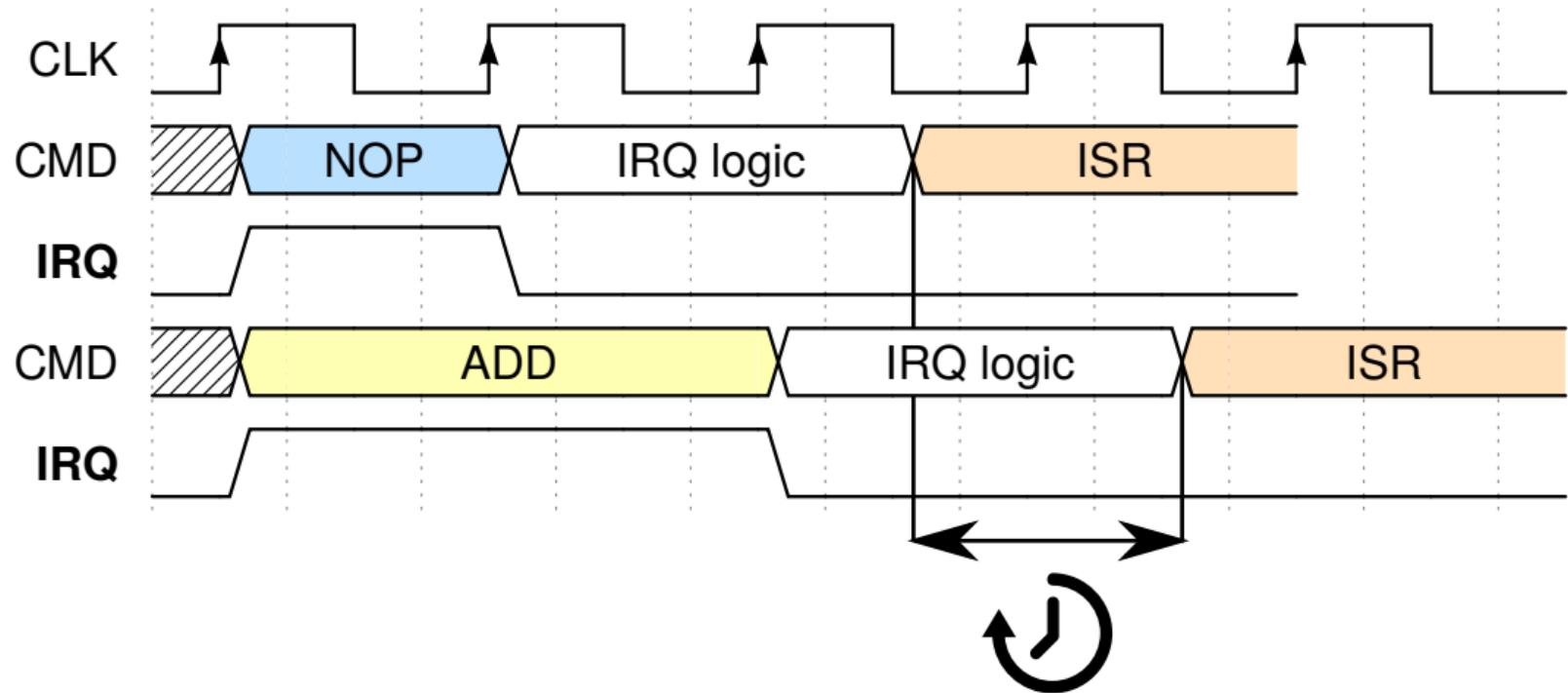
Back to basics: Fetch-decode-execute



Timing leak: IRQ response time depends on current instruction(!)



Wait a cycle: Interrupt latency as a side channel

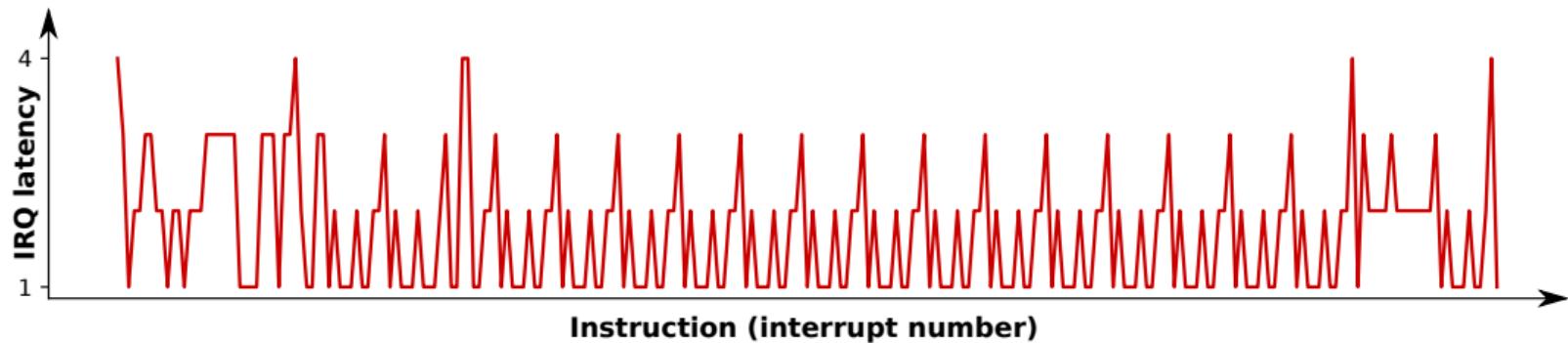


TIMING LEAKS



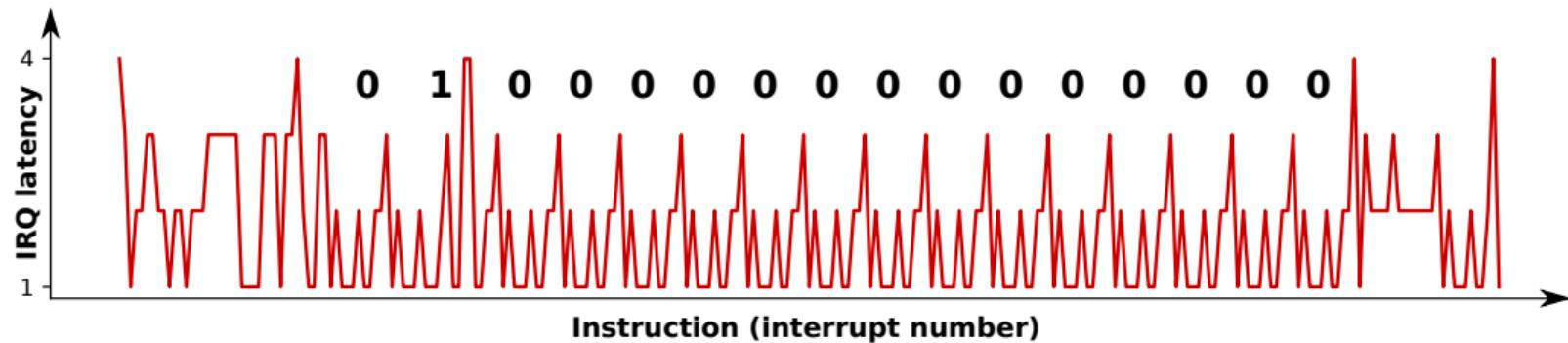
EVERYWHERE

Nemesis attack: Inferring key strokes from Sancus enclaves



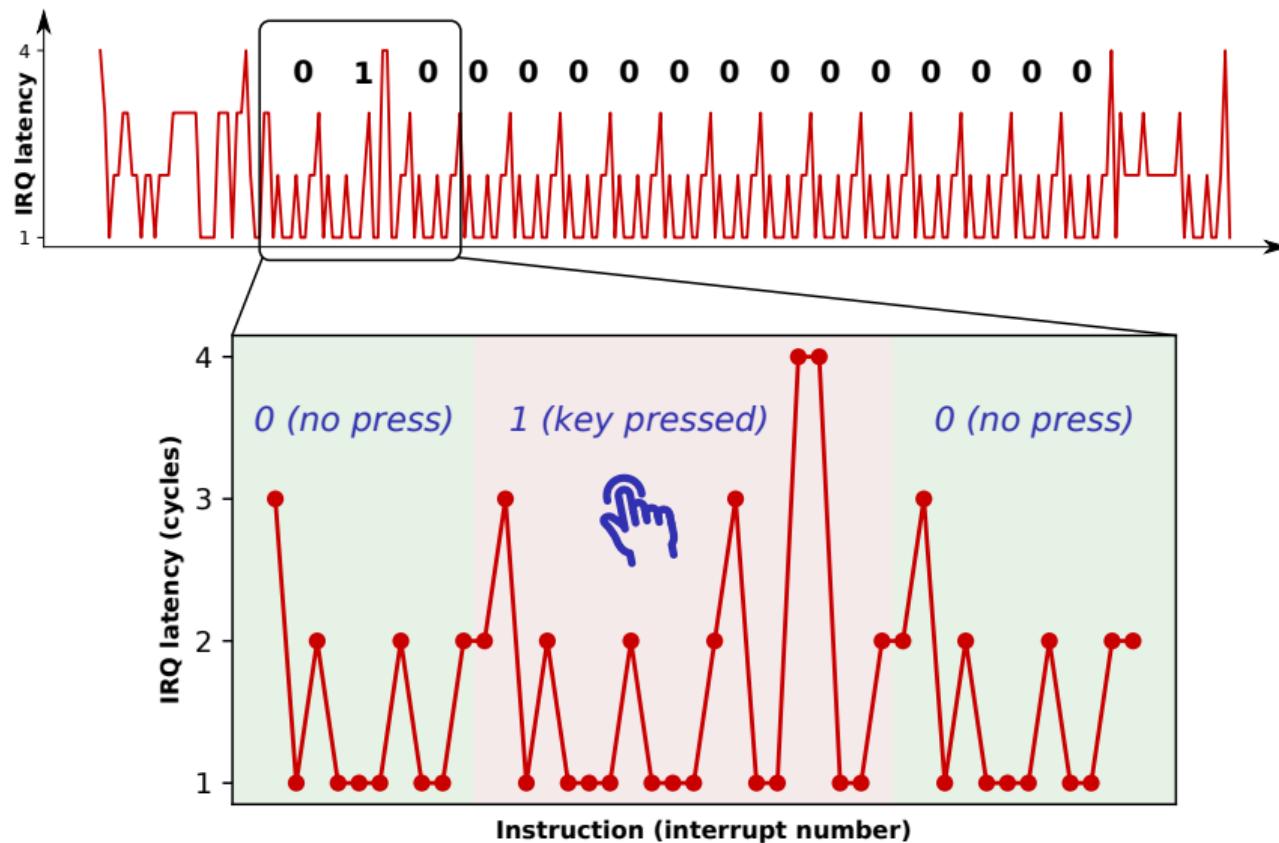
Enclave x-ray: Start-to-end trace enclaved execution

Nemesis attack: Inferring key strokes from Sancus enclaves



Enclave x-ray: Keymap bit traversal (ground truth)

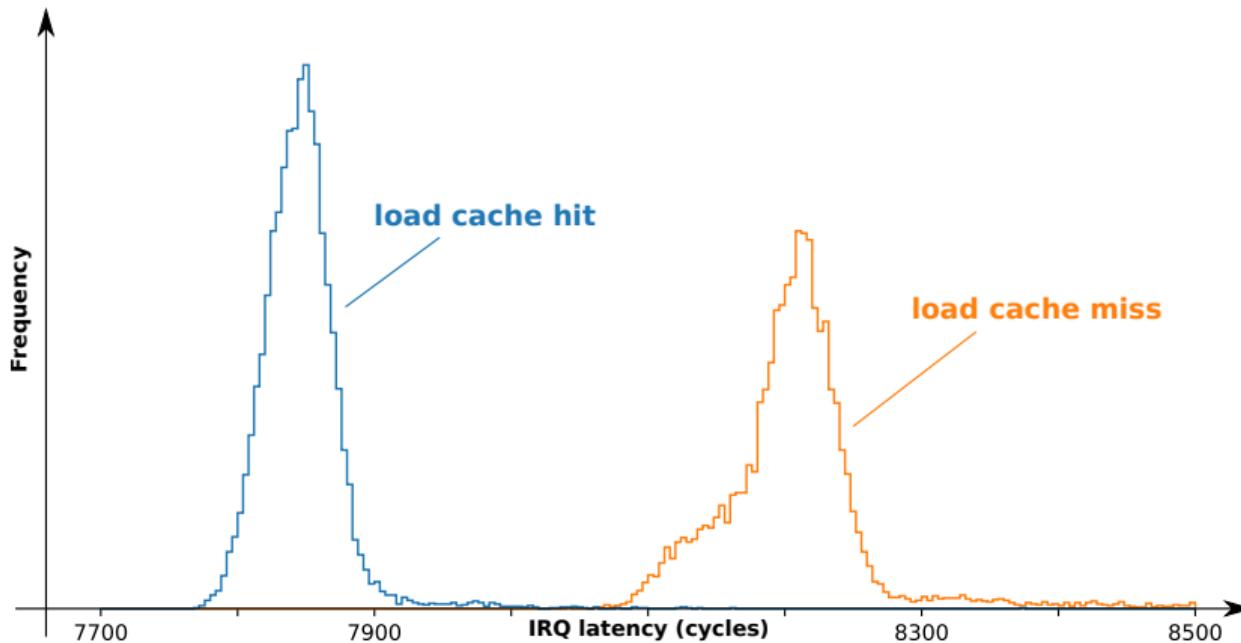
Nemesis attack: Inferring key strokes from Sancus enclaves



Intel SGX microbenchmarks: Measuring x86 cache misses



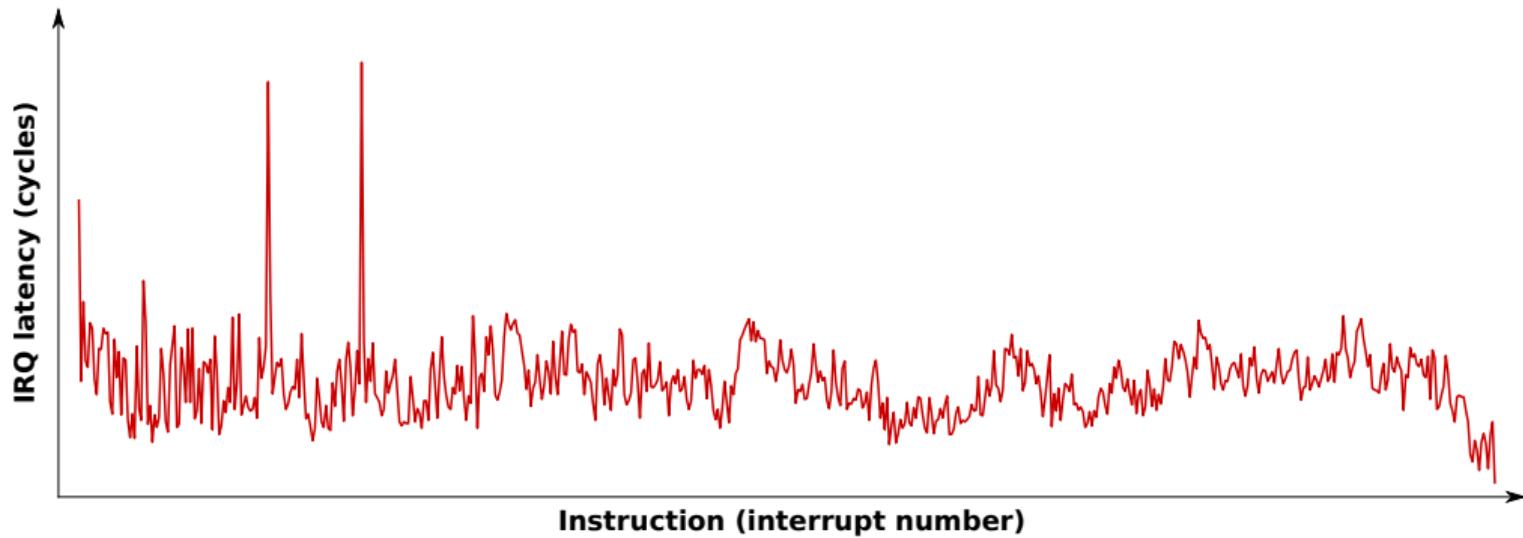
Timing leak: reconstruct *microarchitectural state*



Single-stepping Intel SGX enclaves in practice



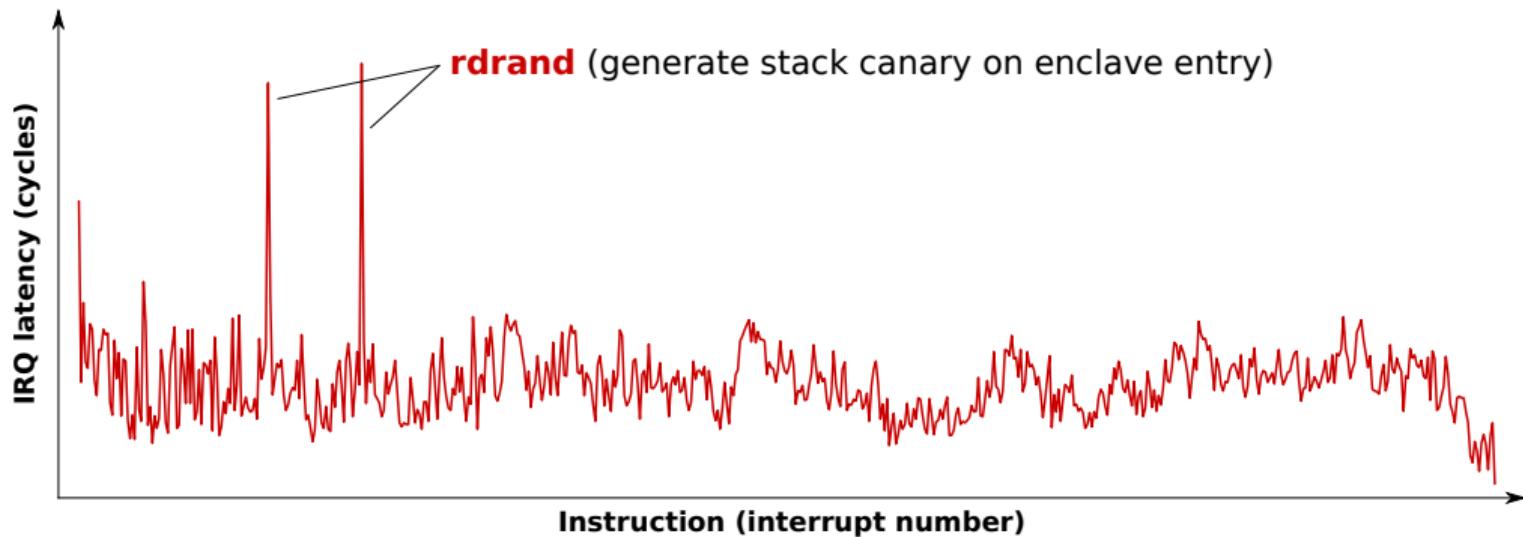
Enclave x-ray: Start-to-end trace enclaved execution



Single-stepping Intel SGX enclaves in practice



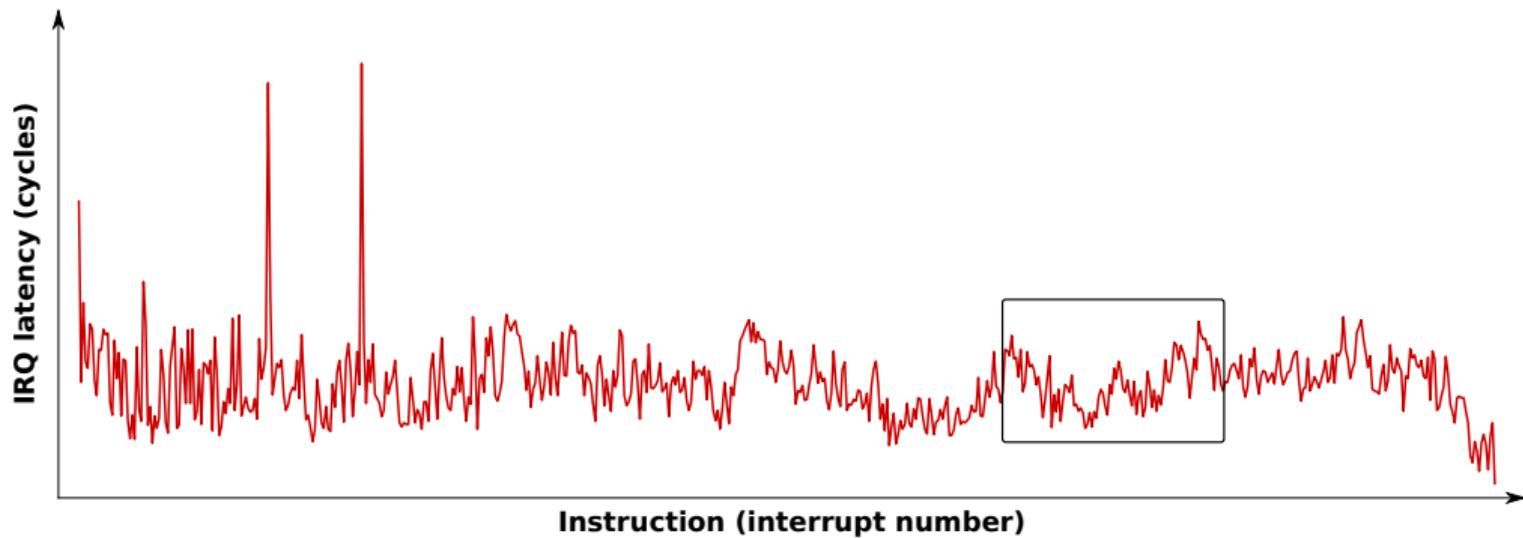
Enclave x-ray: Spotting **high-latency** instructions



Single-stepping Intel SGX enclaves in practice

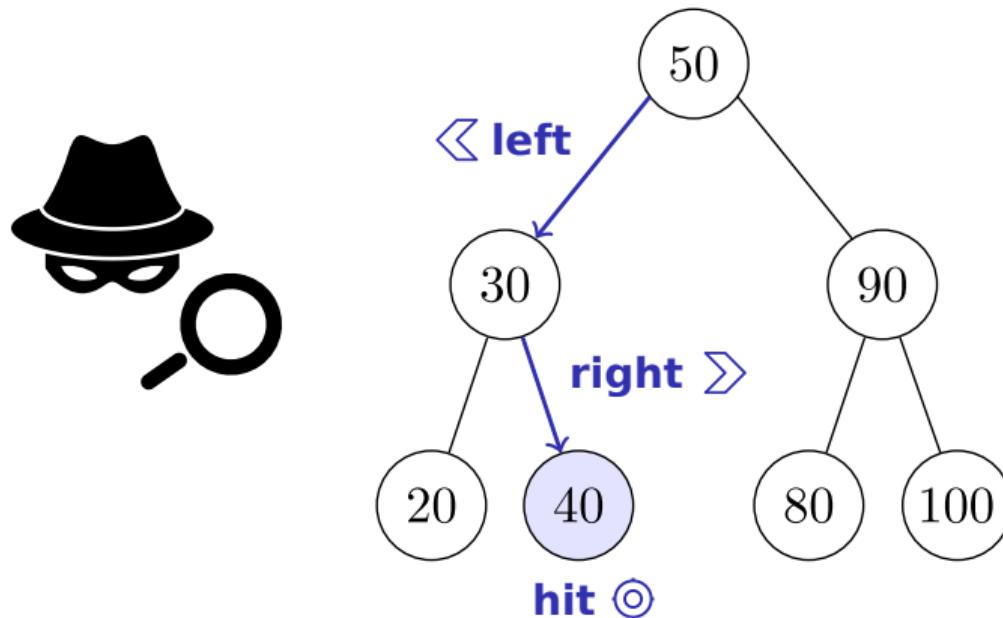


Enclave x-ray: Zooming in on `bsearch` function



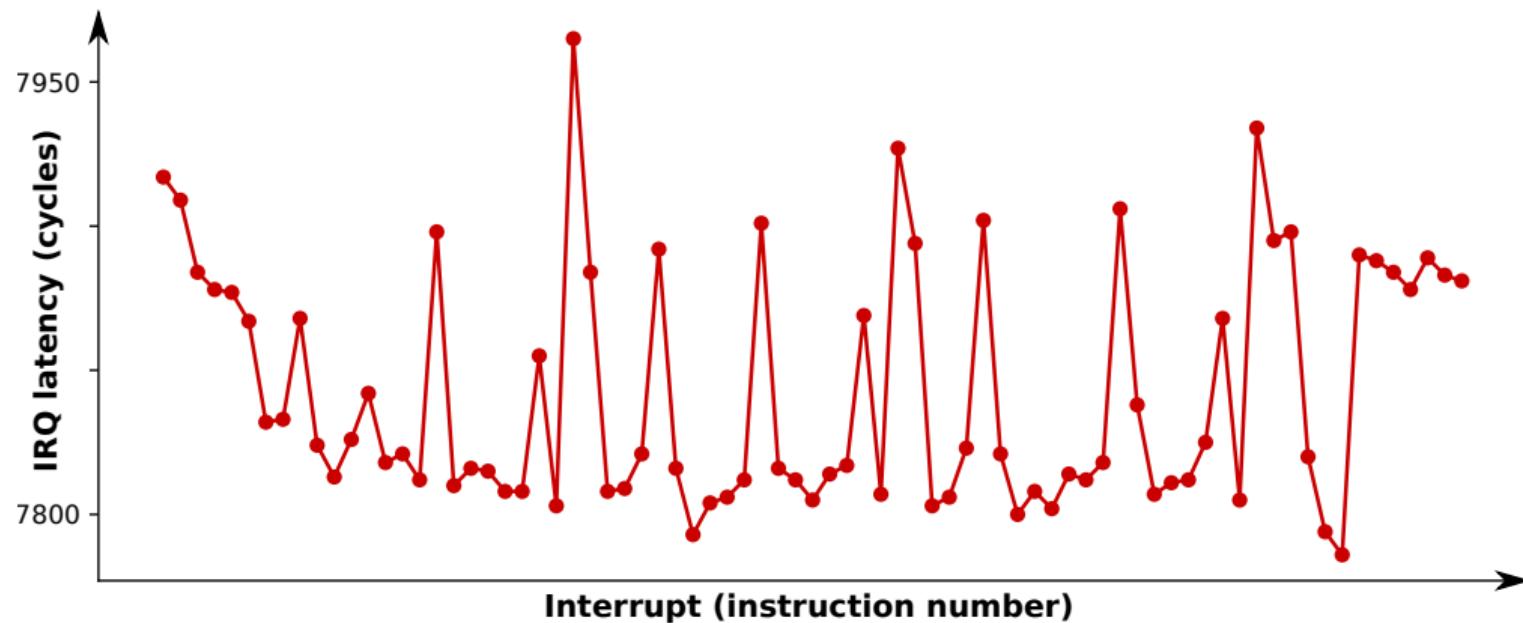
De-anonymizing SGX enclave lookups with interrupt latency

Adversary: Infer **secret lookup** in known sequence (e.g., DNA)



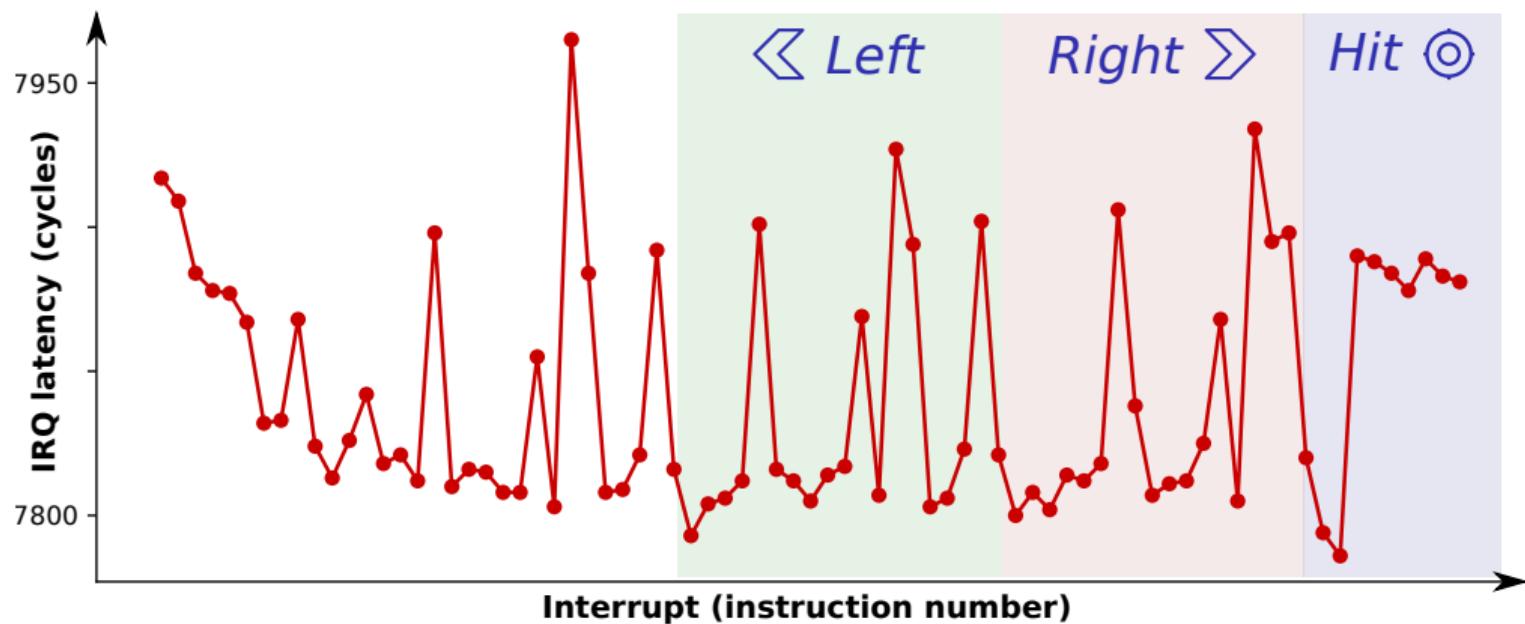
De-anonymizing SGX enclave lookups with interrupt latency

Goal: Infer lookup → reconstruct `bsearch` control flow



De-anonymizing SGX enclave lookups with interrupt latency

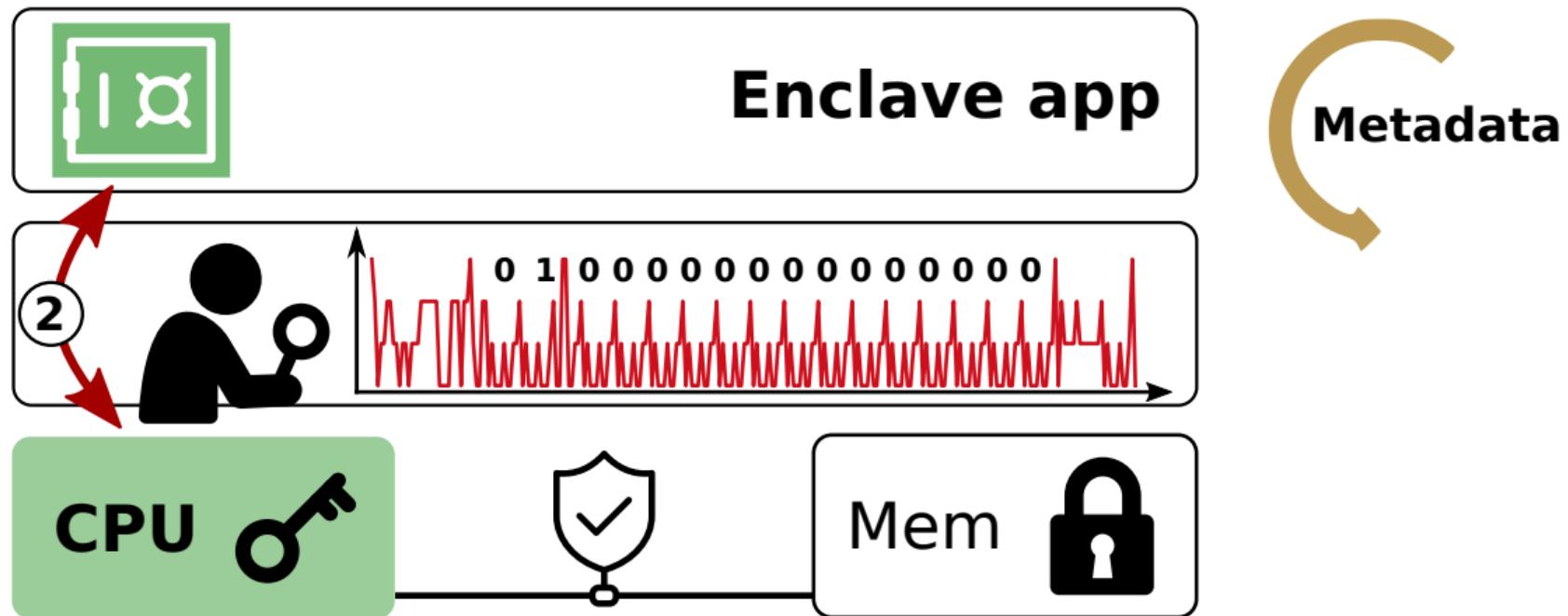
Goal: Infer lookup → reconstruct `bsearch` control flow



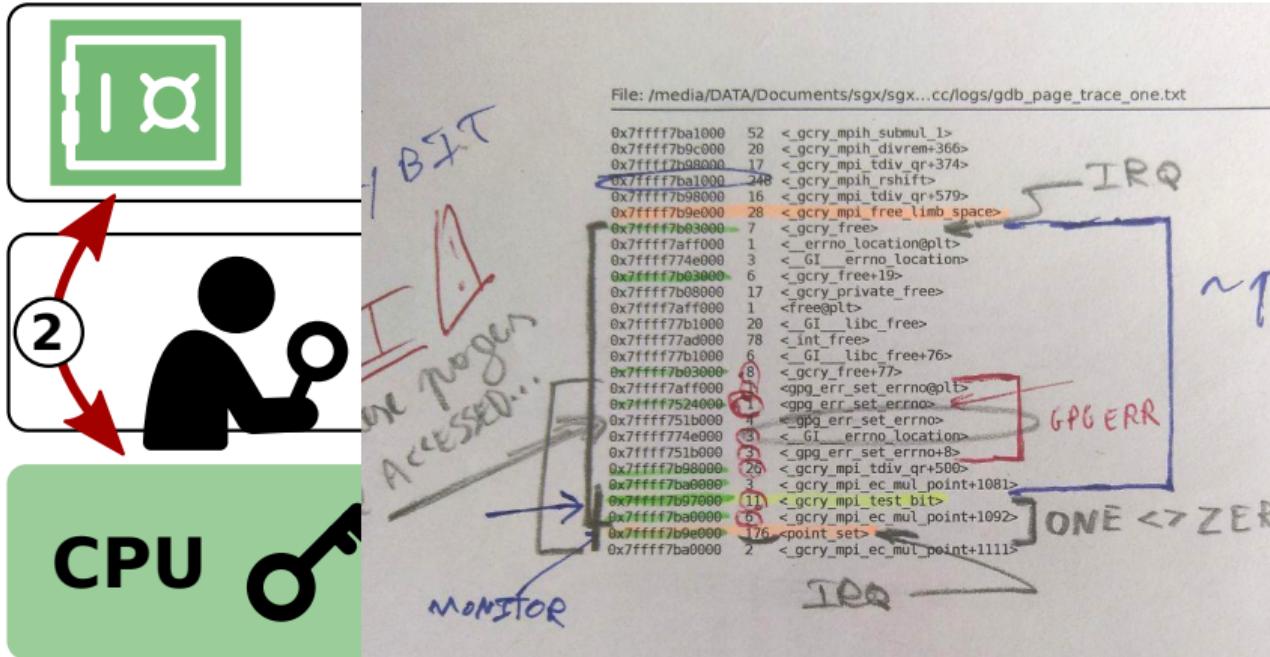


Idea 3: Privileged page tables for transient data leakage

Thesis outline: Privileged side channels (interrupt latency)

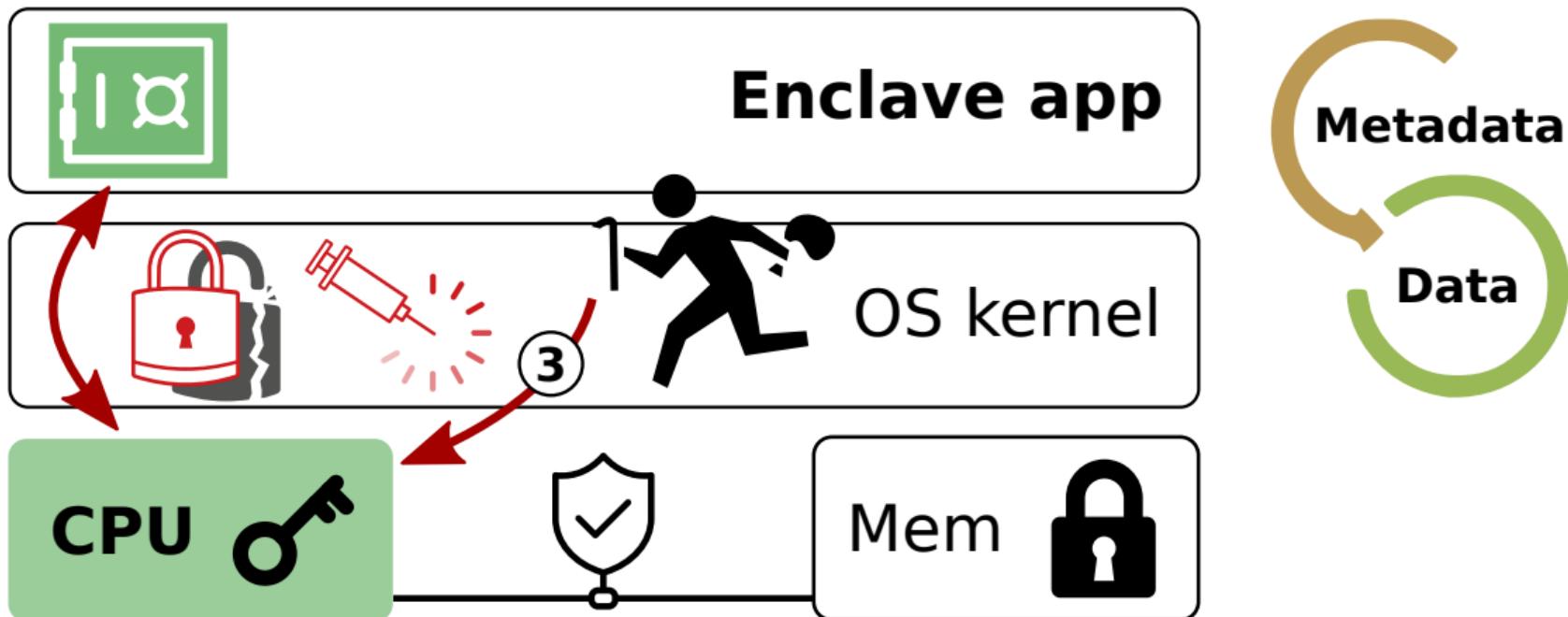


Thesis outline: Privileged side channels (page-table accesses)



Metadata

Thesis outline: Transient-execution attacks (Foreshadow, LVI)



A close-up portrait of Agent Smith from The Matrix. He has his signature bald head, dark sunglasses, and a neutral, slightly smug expression. His skin is pale and textured. The background is a soft-focus green and brown.

WHAT IF I TOLD YOU

YOU CAN CHANGE RULES MID-GAME

THE MELTDOWN AND SPECTRE EXPLOITS USE
"SPECULATIVE EXECUTION?" WHAT'S THAT?

YOU KNOW THE TROLLEY PROBLEM? WELL,
FOR A WHILE NOW, CPUs HAVE BASICALLY
BEEN SENDING TROLLEYS DOWN BOTH
PATHS, QUANTUM-STYLE, WHILE AWAITING
YOUR CHOICE. THEN THE UNNEEDED
"PHANTOM" TROLLEY DISAPPEARS.



THE PHANTOM TROLLEY ISN'T
SUPPOSED TO TOUCH ANYONE.
BUT IT TURNS OUT YOU CAN
STILL USE IT TO DO STUFF.

AND IT CAN DRIVE
THROUGH WALLS.



THE MELTDOWN AND SPECTRE EXPLOITS USE
"SPECULATIVE EXECUTION?" WHAT'S THAT?

YOU KNOW THE TROLLEY PROBLEM? WELL,
FOR A WHILE NOW CPUs HAVE RADICALLY

Key finding of 2018

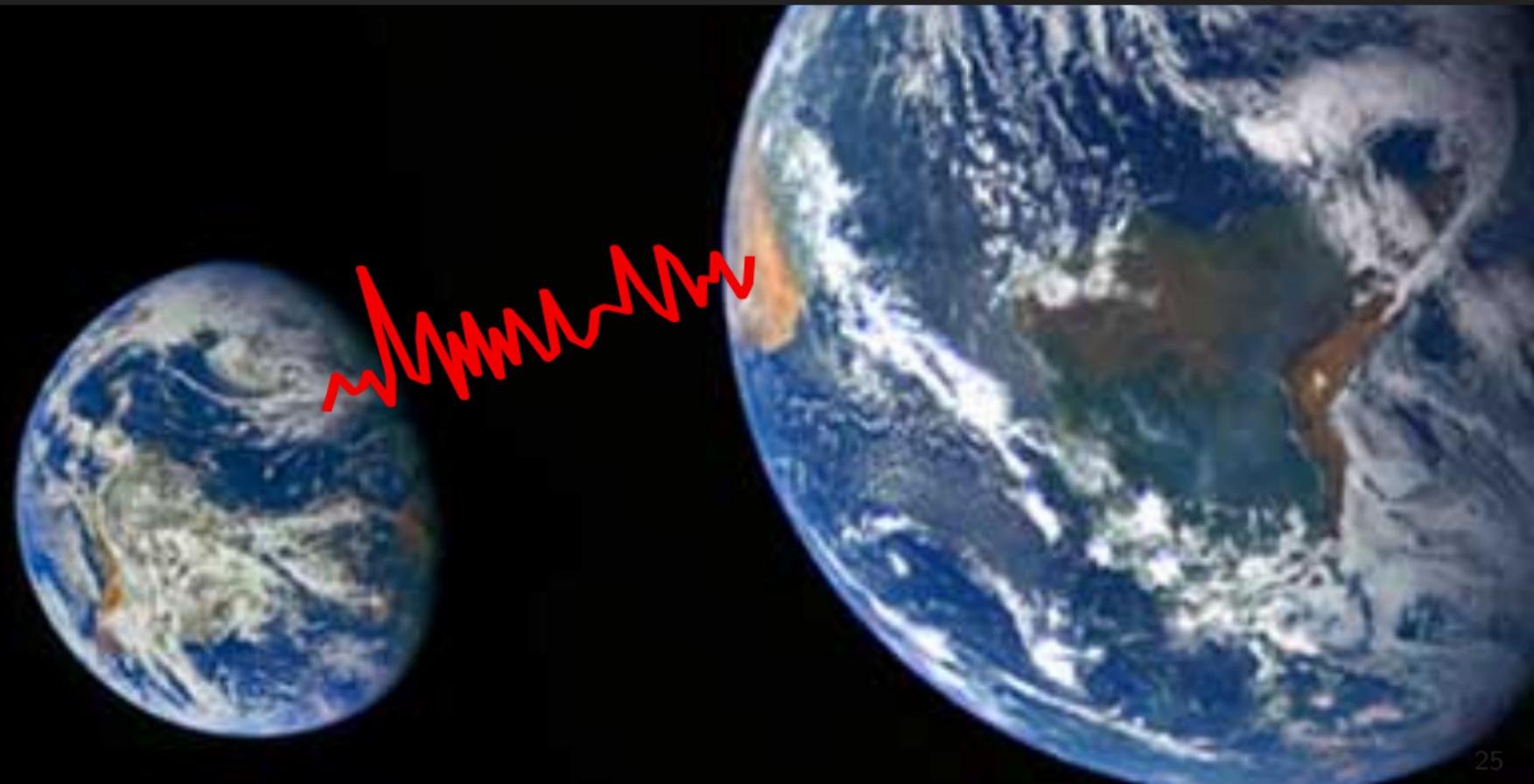
- CPU executes ahead of time in **transient world**
- Use **side channels** to reconstruct secrets!

THE PHANTOM TROLLEY ISN'T
SUPPOSED TO TOUCH ANYONE.
BUT IT TURNS OUT YOU CAN
STILL USE IT TO DO STUFF.



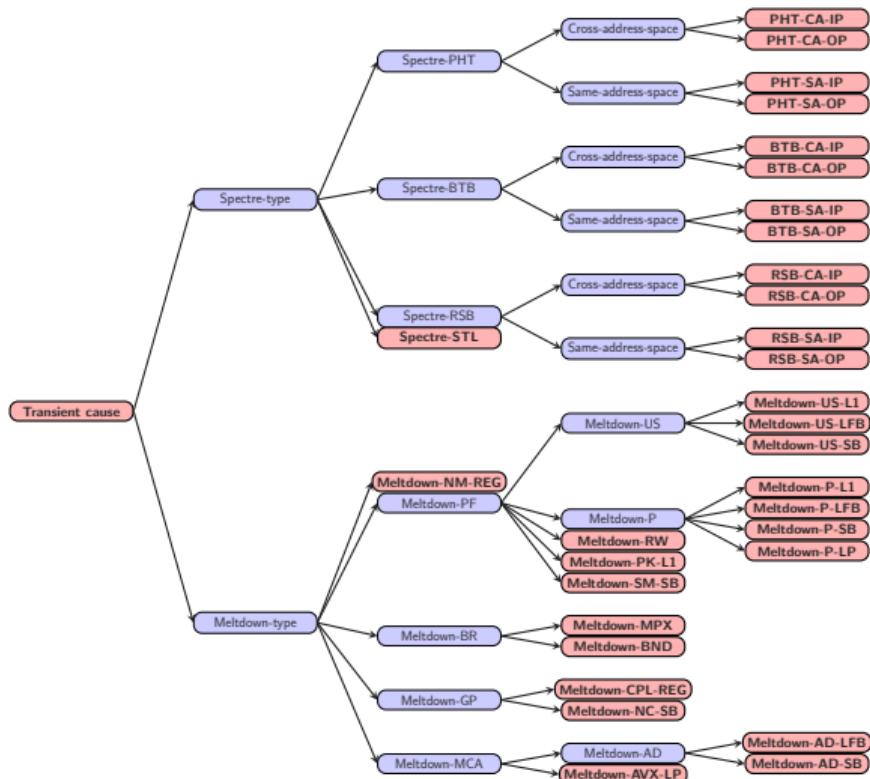


Transient-execution attacks: Welcome to the world of fun!



The transient-execution zoo

<https://transient.fail>





inside™

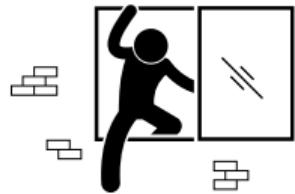


inside™



inside™

Meltdown: Transiently encoding unauthorized memory



Unauthorized access

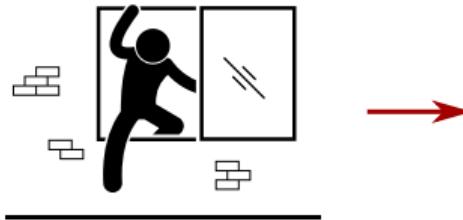
Listing 1: x86 assembly

```
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

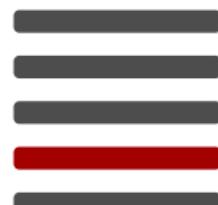
Listing 1: x86 assembly.

```
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.

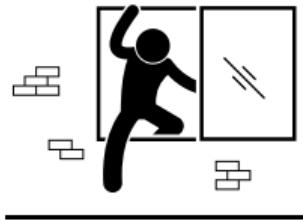
```
1 void meltdown(  
2     uint8_t *oracle,  
3     uint8_t *secret_ptr)  
4 {  
5     uint8_t v = *secret_ptr;  
6     v = v * 0x1000;  
7     uint64_t o = oracle[v];  
8 }
```

oracle array



secret idx

Meltdown: Transiently encoding unauthorized memory



Unauthorized access



Transient out-of-order window



Exception

(discard architectural state)

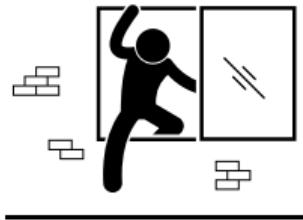
Listing 1: x86 assembly.

```
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

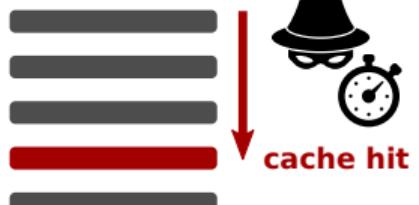
Listing 1: x86 assembly.

```
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 }
```

oracle array





inside™



inside™



inside™

Rumors: Meltdown immunity for SGX enclaves?

Meltdown melted down everything, except for one thing

“[enclaves] remain **protected and completely secure**”

— *International Business Times, February 2018*

ANJUNA'S SECURE-RUNTIME CAN PROTECT CRITICAL APPLICATIONS AGAINST THE MELTDOWN ATTACK USING ENCLAVES

“[enclave memory accesses] redirected to an **abort page**, which has no value”

— *Anjuna Security, Inc., March 2018*

Rumors: Meltdown immunity for SGX enclaves?



LILY HAY NEWMAN SECURITY 08.14.18 01:00 PM

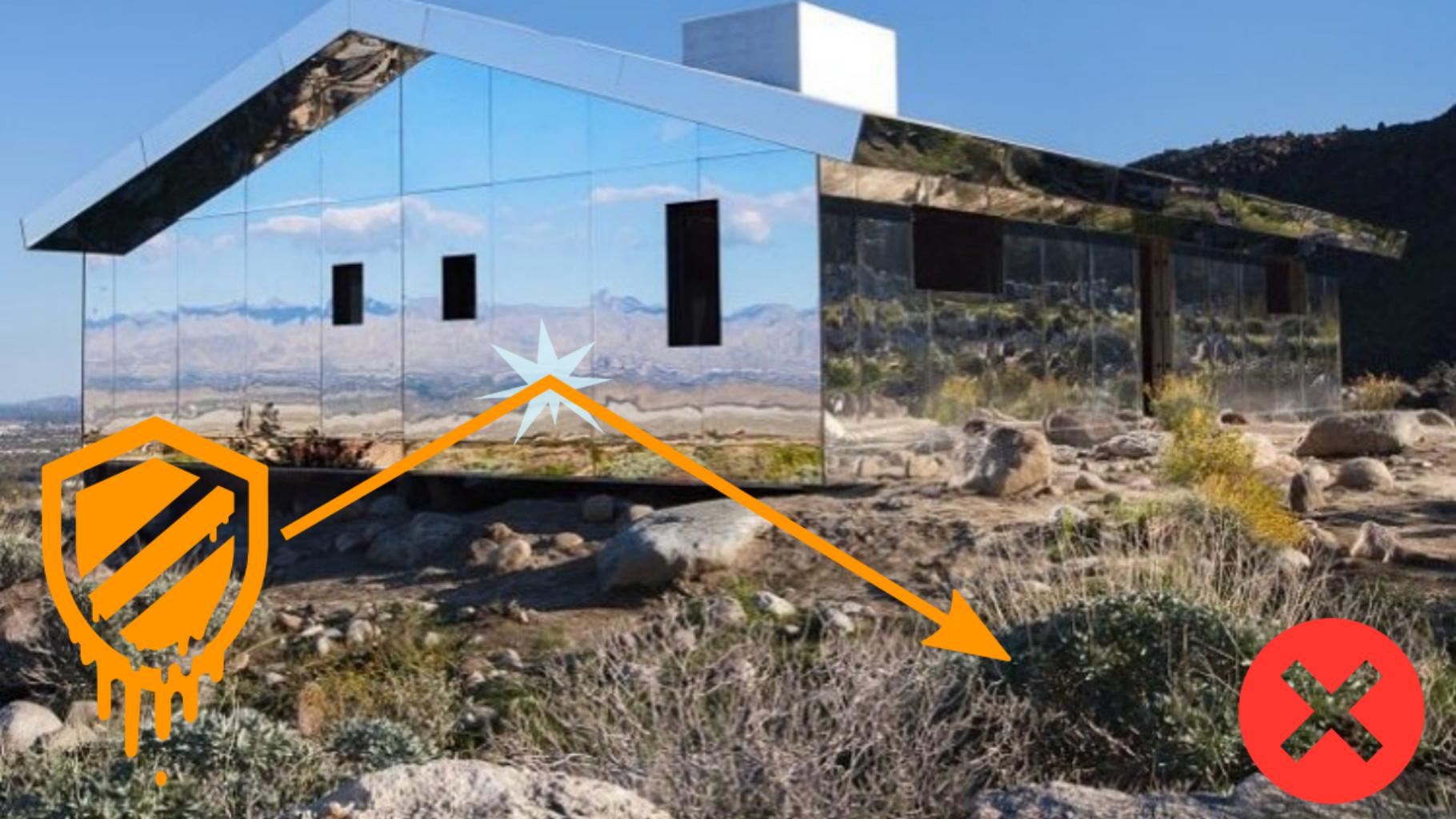
SPECTRE-LIKE FLAW UNDERMINES INTEL PROCESSORS' MOST SECURE ELEMENT

I'M SURE THIS WON'T BE THE LAST SUCH PROBLEM —

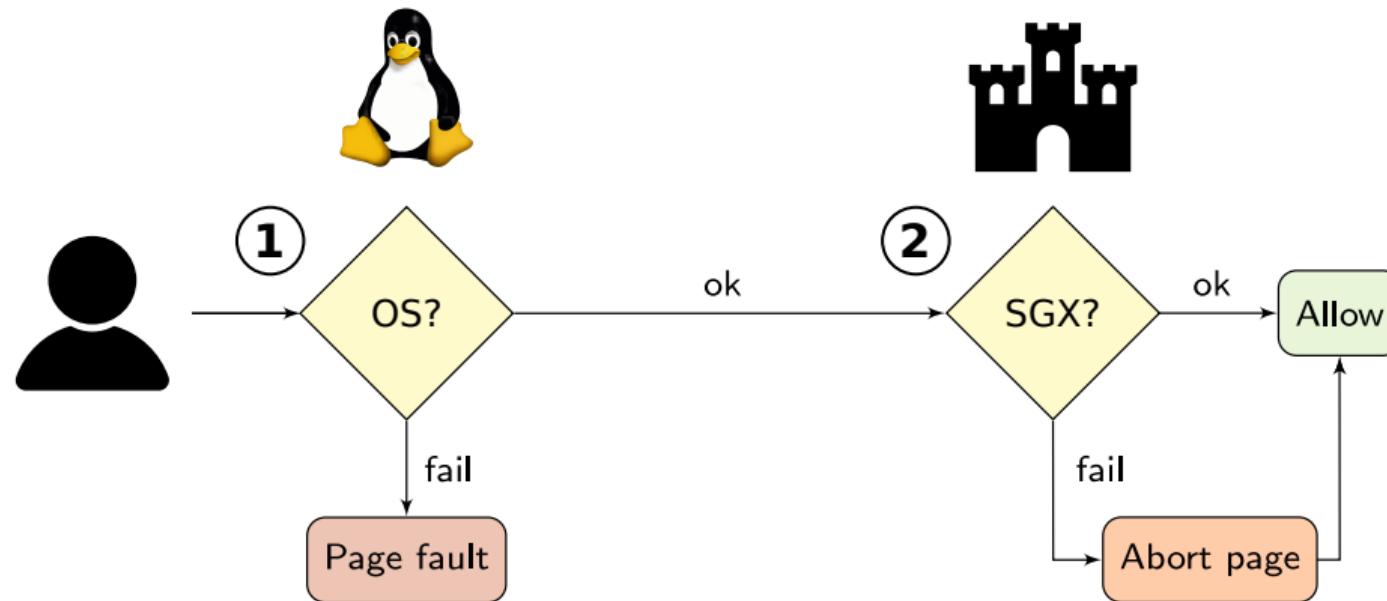
Intel's SGX blown wide open by, you guessed it, a speculative execution attack

Speculative execution attacks truly are the gift that keeps on giving.

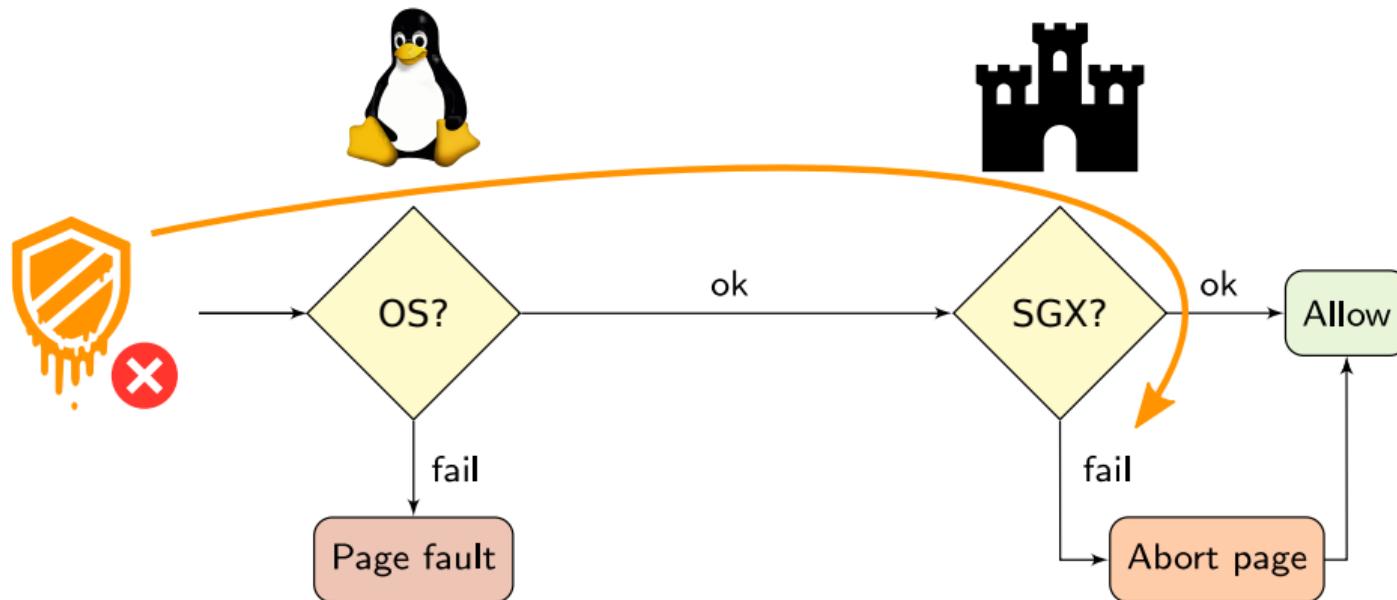




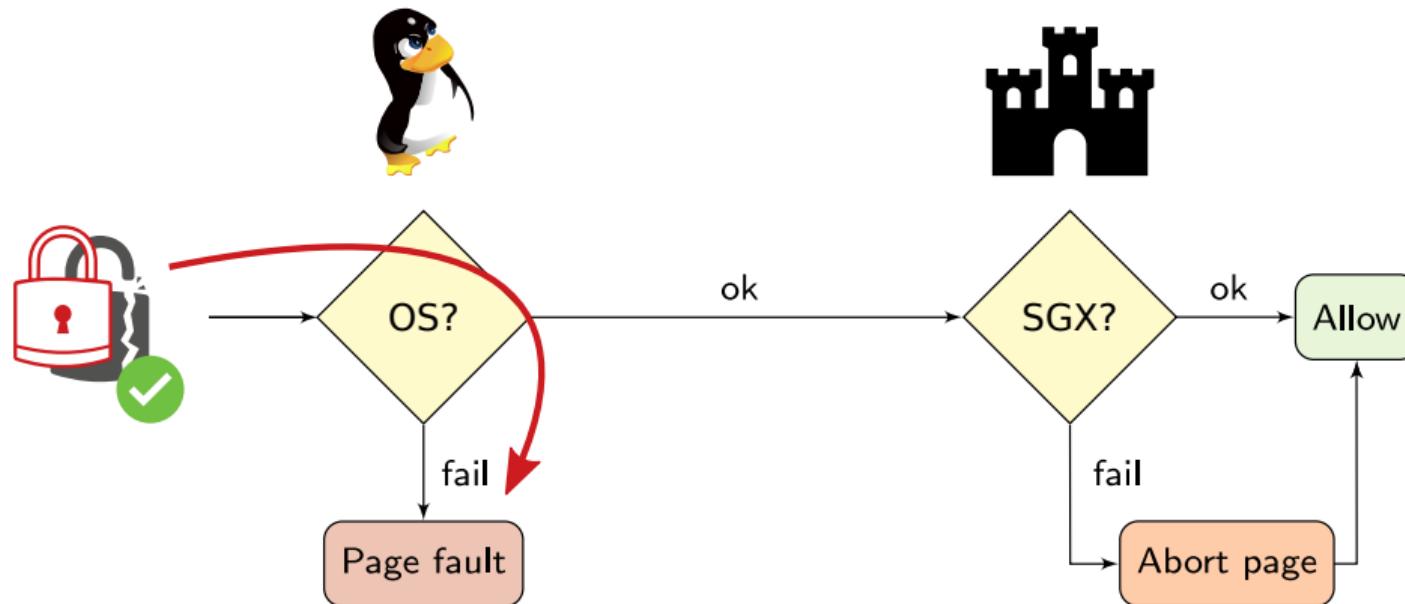
Building Foreshadow: Evade SGX abort page semantics



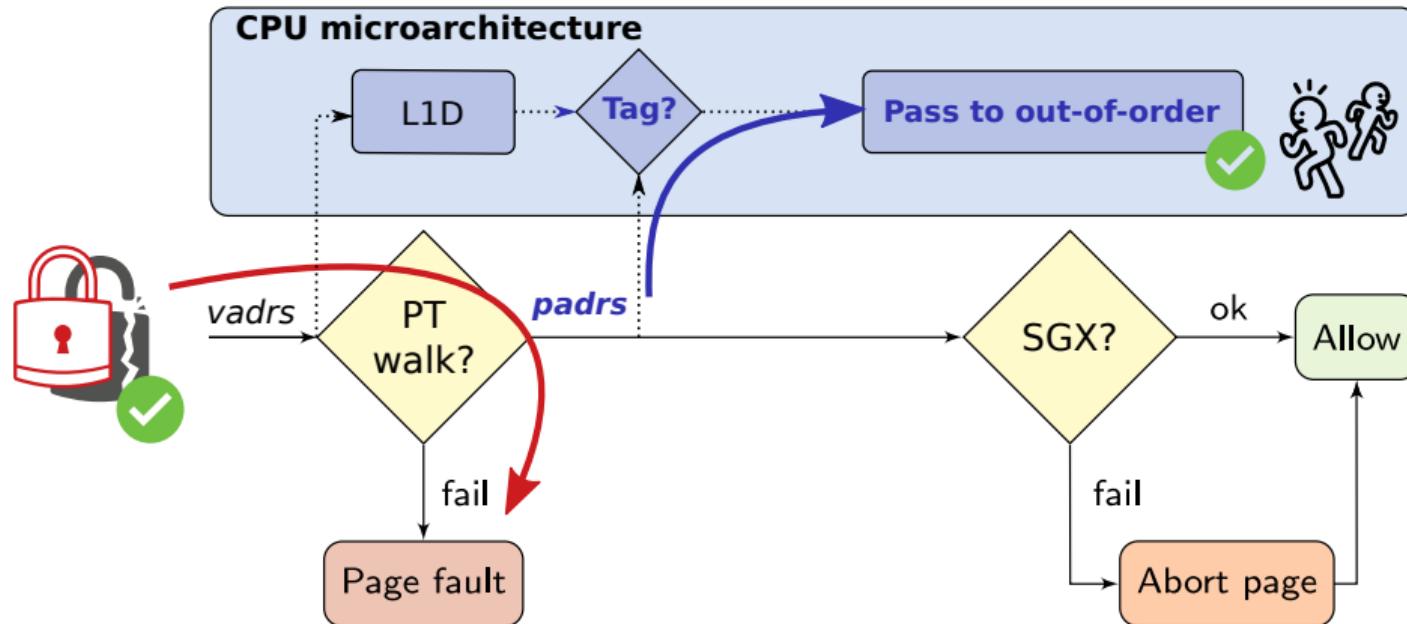
Building Foreshadow: Evade SGX abort page semantics



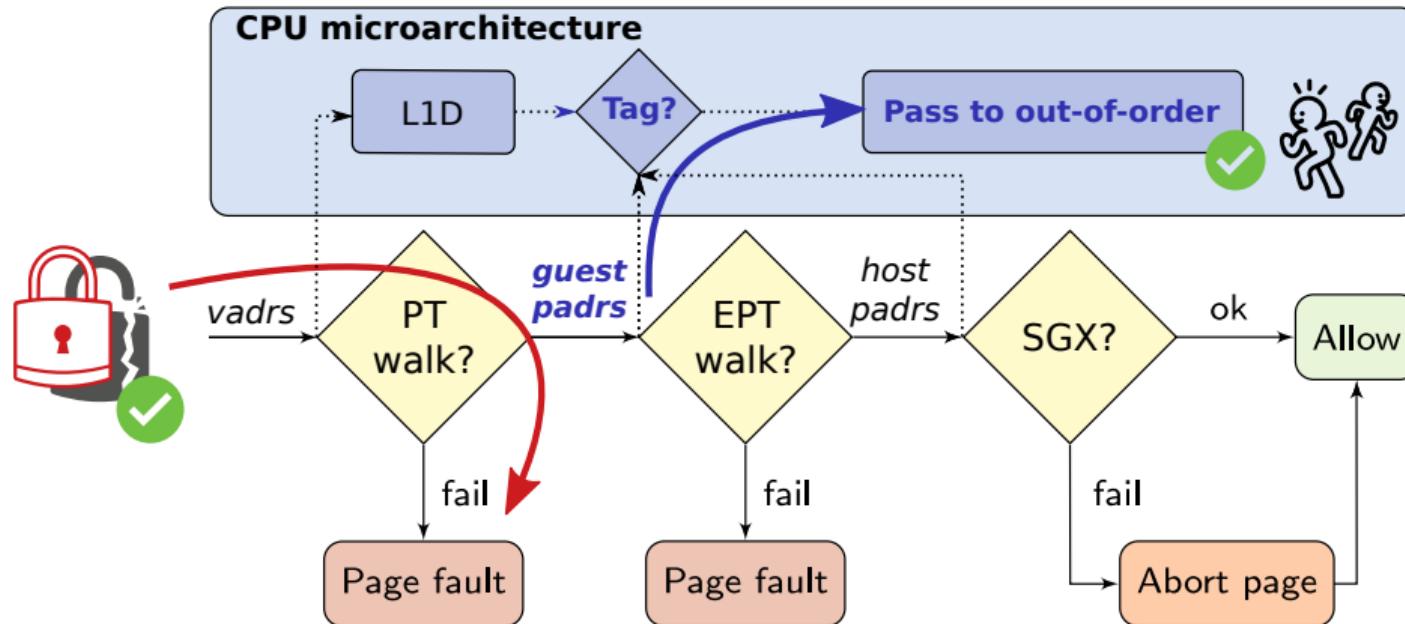
Building Foreshadow: Evade SGX abort page semantics



Foreshadow-SGX: Breaking enclave isolation



Foreshadow-NG: Breaking virtual machine isolation



Terminal

Foreshadow Demo

SGX enclave: secret string at 0x7f19ee646000

Press enter to naively read enclave memory at address 0x7f19ee646000...

Segment 0: 0x7f19ee646000 - 0x7f19ee646317

Victim address = 0x7f19ee646316... 0xFF

Actual success rate = 0/791 = 0.00 %

Press enter to use Foreshadow to read enclave memory at address 0x7f19ee646000 ...

Segment 0: 0x7f19ee646000 - 0x7f19ee646317

Victim address = 0x7f19ee6460dd... 0x69

Extracted Bytes -

49 74 20 77 61 73 2

However FORESHADOW are so contrived that the eyes follow you about when you move. BIG BROTHER IS WATCHING YOU, the caption beneath it ran. Inside the flat a fruity voice was reading out a list of figures w.....

However FORESHADOW

can read the actual enclave memory



Mitigating Foreshadow: Flush CPU microarchitecture



Mitigating Foreshadow: Flush CPU microarchitecture



10BH	267	IA32_FLUSH_CMD	Flush Command (W0) Gives software a way to invalidate structures with finer granularity than other architectural methods.	If any one of the enumeration conditions for defined bit field positions holds.
	0		L1D_FLUSH: Writeback and invalidate the L1 data cache.	If CPUID.(EAX=07H, ECX=0):EDX[28]=1
	63:1		Reserved	



inside™

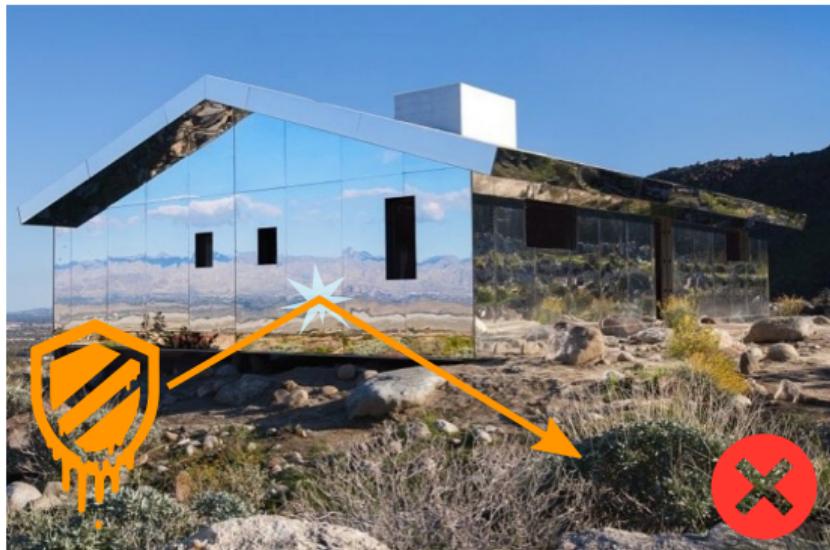


inside™



inside™

Idea: Can we turn Foreshadow around?



Outside view

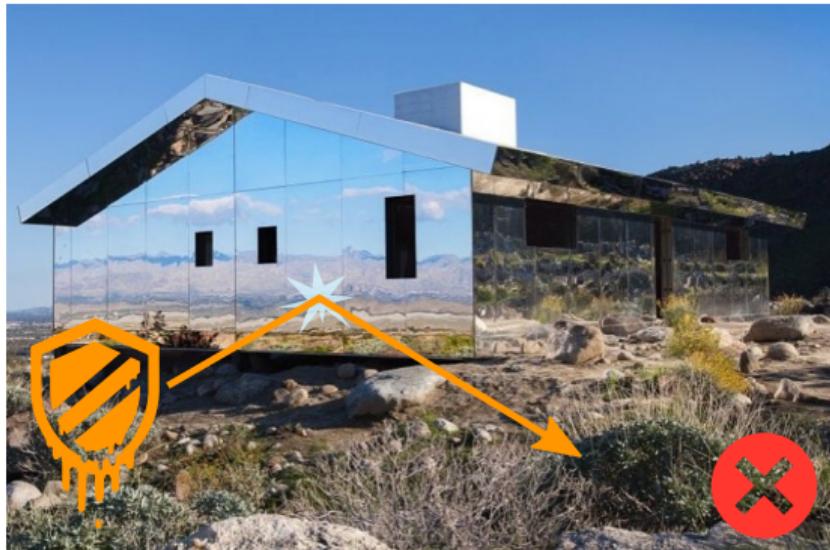
- Meltdown: out-of-reach
- Foreshadow: cache emptied



Intra-enclave view

- Access enclave + outside memory

Idea: Can we turn Foreshadow around?



Outside view

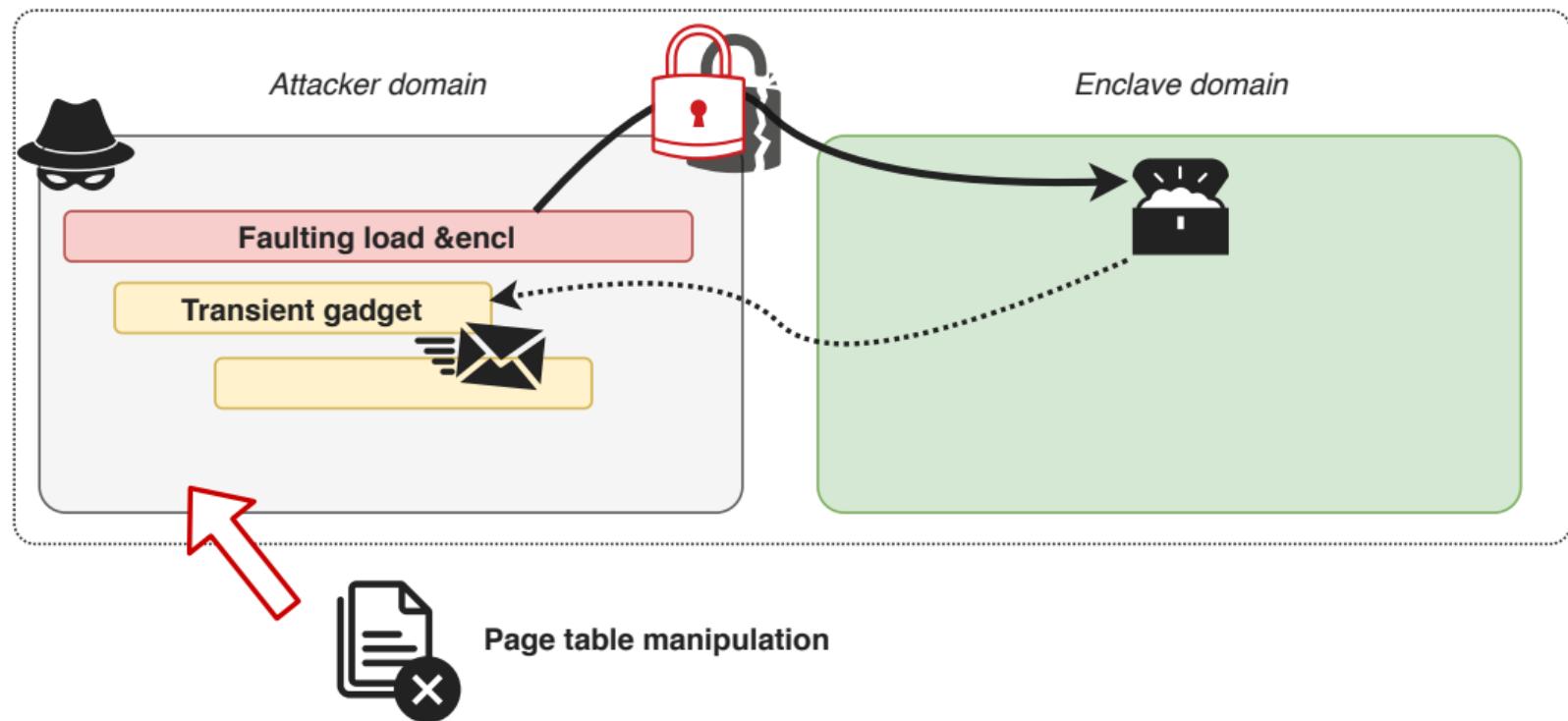
- Meltdown: out-of-reach
- Foreshadow: cache emptied



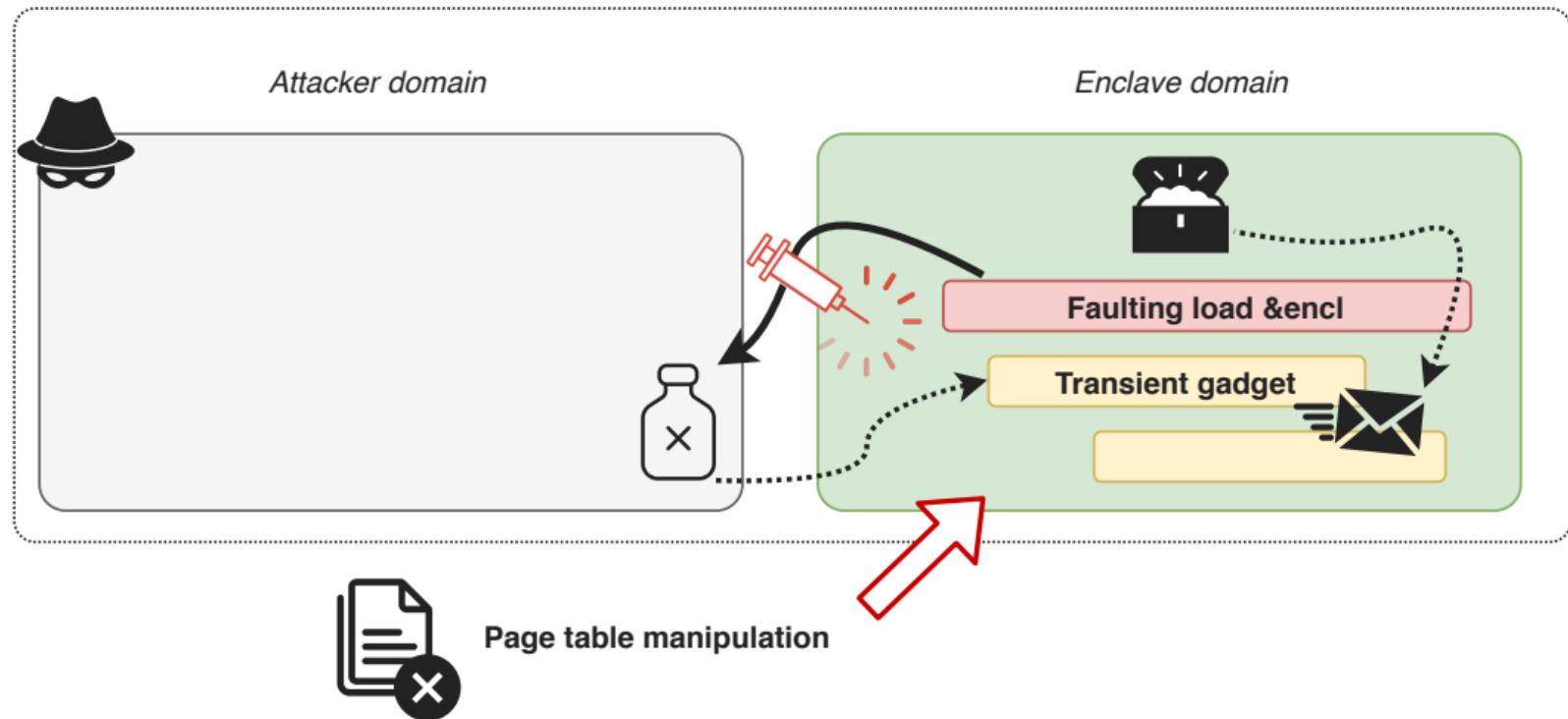
Intra-enclave view

- Access enclave + outside memory
- Abuse **in-enclave code gadgets!**

Reviving Foreshadow & co. with Load Value Injection (LVI)



Reviving Foreshadow & co. with Load Value Injection (LVI)



FOOD POISONING



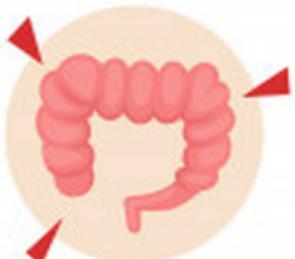
Overdue products



Medicine



Dizziness



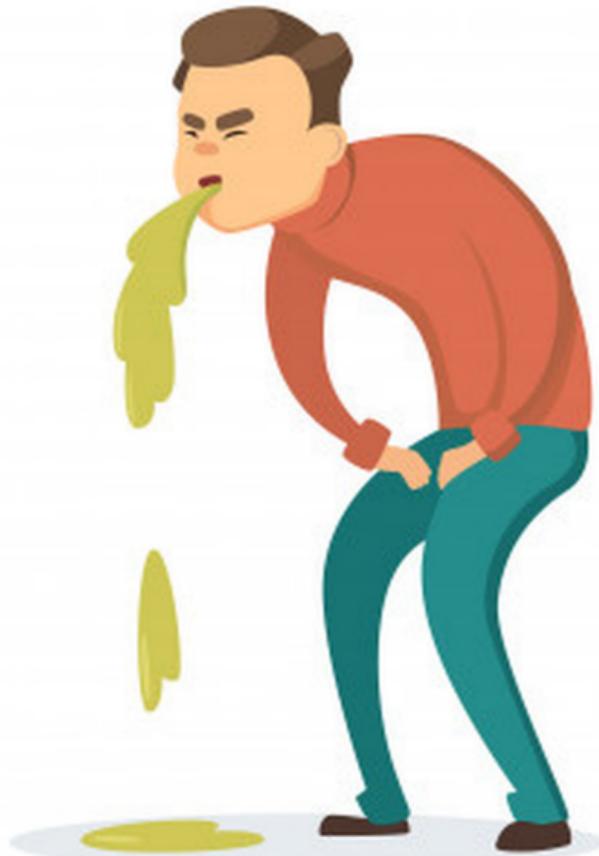
Intestinal colic



Diarrhea



Headache



```
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
```

Mitigating LVI: Fencing vulnerable load instructions



Mitigating LVI: Fencing vulnerable load instructions



LFENCE—Load Fence

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
NP OF AE E8	LFENCE	Z0	Valid	Valid	Serializes load operations.

A smaller red rectangular sign with the words "ALL WAY" written in white capital letters. It is mounted on a post and is positioned in front of a yellow house and some trees.

Mitigating LVI: Compiler and assembler support



-mlfence-after-load

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](#)



-mlvi-hardening

LLVM Lands Performance-Hitting Mitigation For Intel LVI Vulnerability

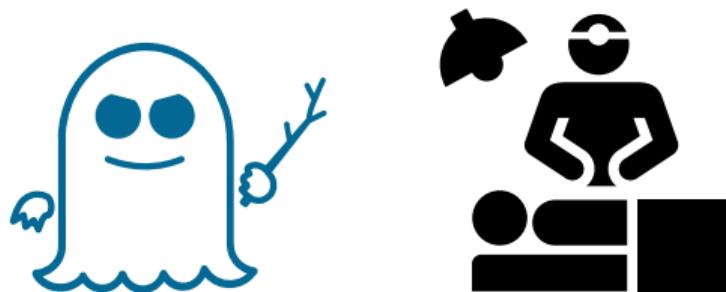
Written by Michael Larabel in [Software](#) on 3 April 2020. [Page 1 of 3](#). [20 Comments](#)



-Qspectre-load

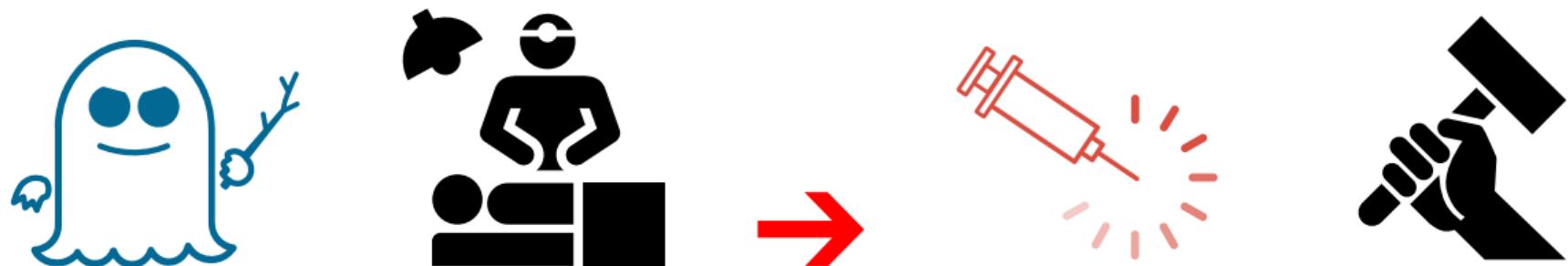
More Spectre Mitigations in [MSVC](#)

March 13th, 2020



23 fences

October 2019—“surgical precision”



23 fences

October 2019—“surgical precision”

49,315 fences

March 2020—“big hammer”



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

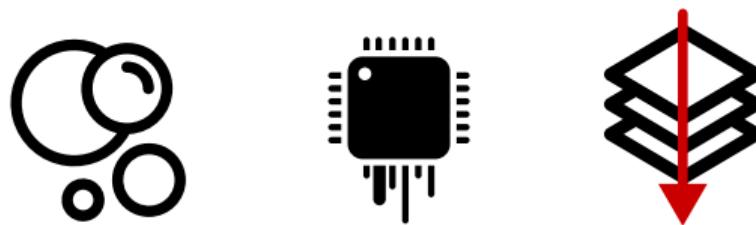
Written by Michael Larabel in [Software](#) on 3 April 2020. [Page 1 of 3](#). [20 Comments](#)

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](#)

Conclusions and takeaway

- ⇒ Trusted execution environments (Intel SGX) ≠ perfect(!)
- ⇒ Importance of fundamental side-channel research; no silver-bullet defenses
- ⇒ Security cross-cuts the system stack: hardware, OS, compiler, application





Thank you!