



Privileged Side-Channel Attacks

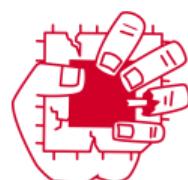
for Enclave Software Adversaries

Jo Van Bulck

University of Birmingham seminar, February 20, 2020

🏡 imec-DistriNet, KU Leuven 📩 jo.vanbulck@cs.kuleuven.be 🐦 jovanbulck

- Trusted computing **across the system stack**: hardware, compiler, OS, apps
- Integrated **attack-defense** perspective and **open-source** prototypes



Transient execution

[VBMW⁺18, SLM⁺19, CVBS⁺18]

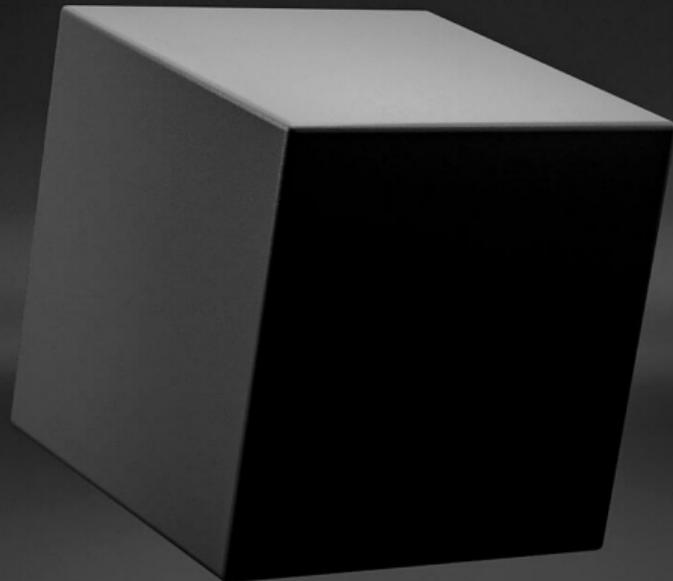
Side-channel attacks

[VBPS17, VBWK⁺17, VBPS18]

Sancus TEE processor

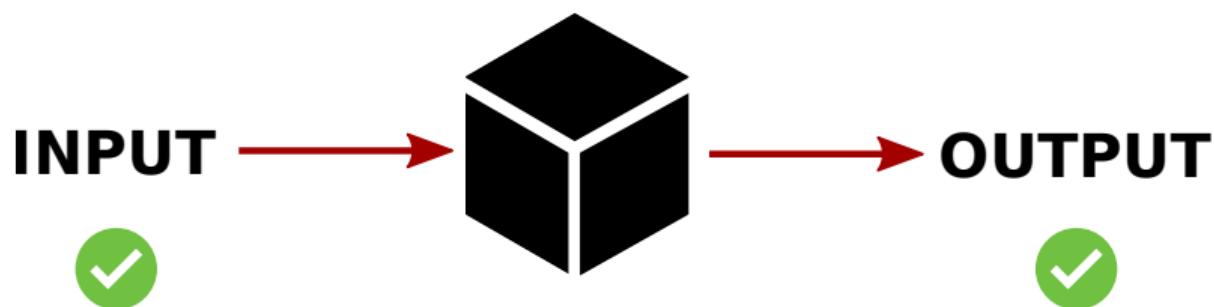
[NVBM⁺17, VBMP17]

~40 years of computer security research in one picture



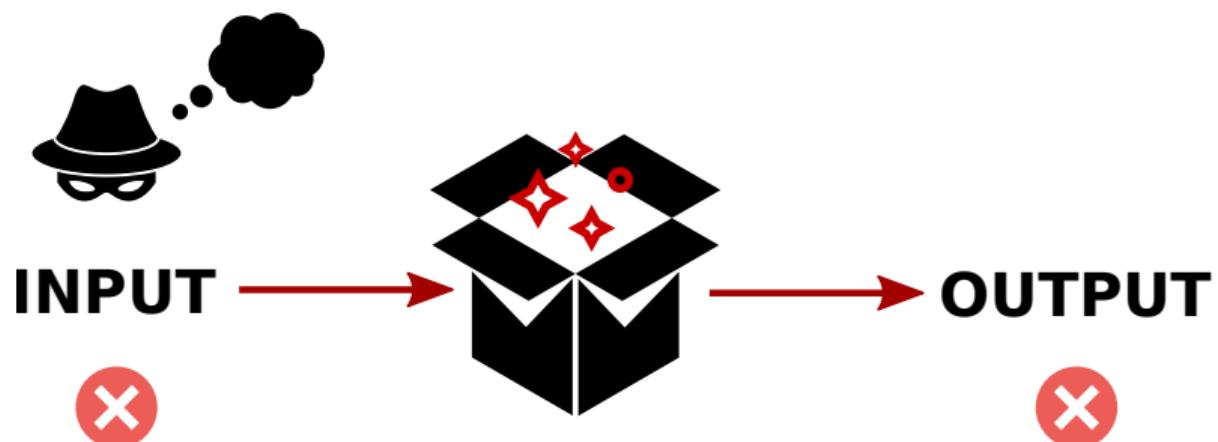
A primer on software security

Secure program: convert all input to *expected output*



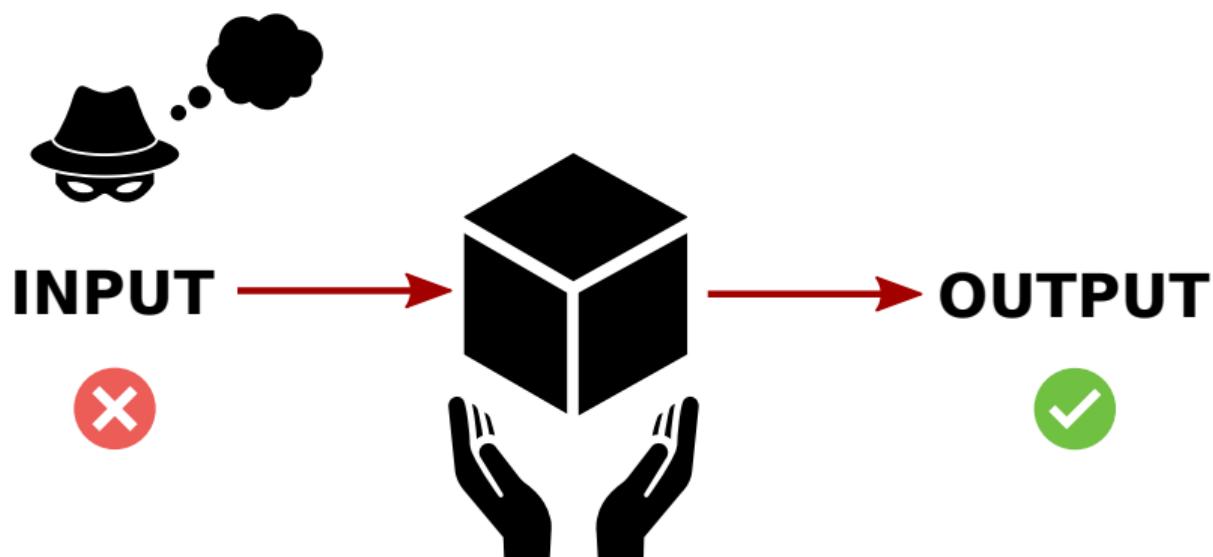
A primer on software security

Buffer overflow vulnerabilities: trigger *unexpected behavior*



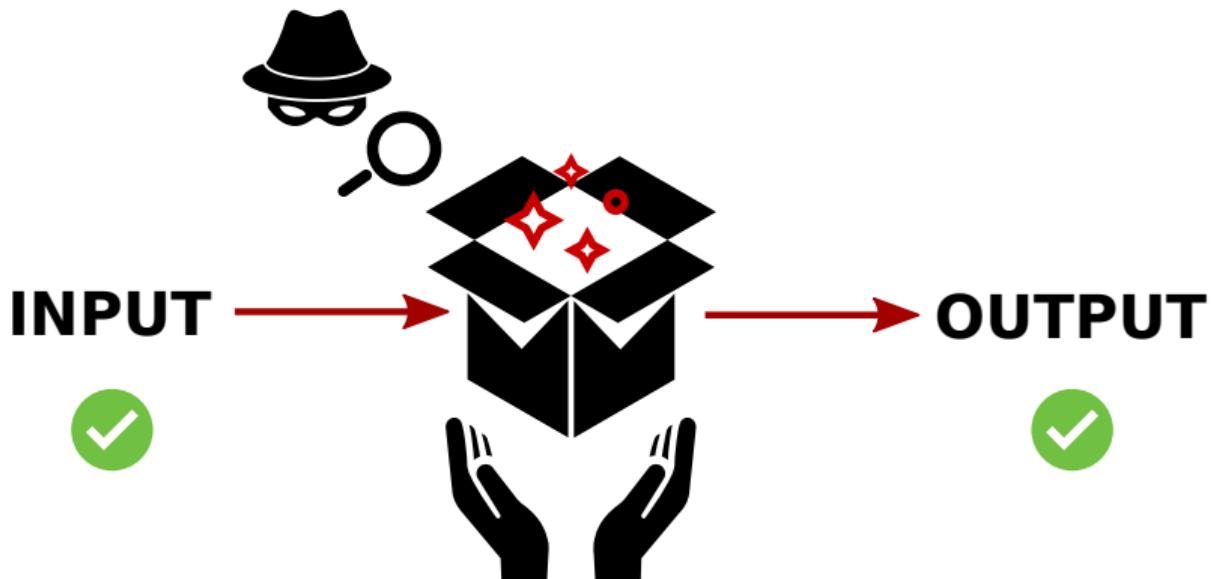
A primer on software security

Safe languages & formal verification: preserve expected behavior



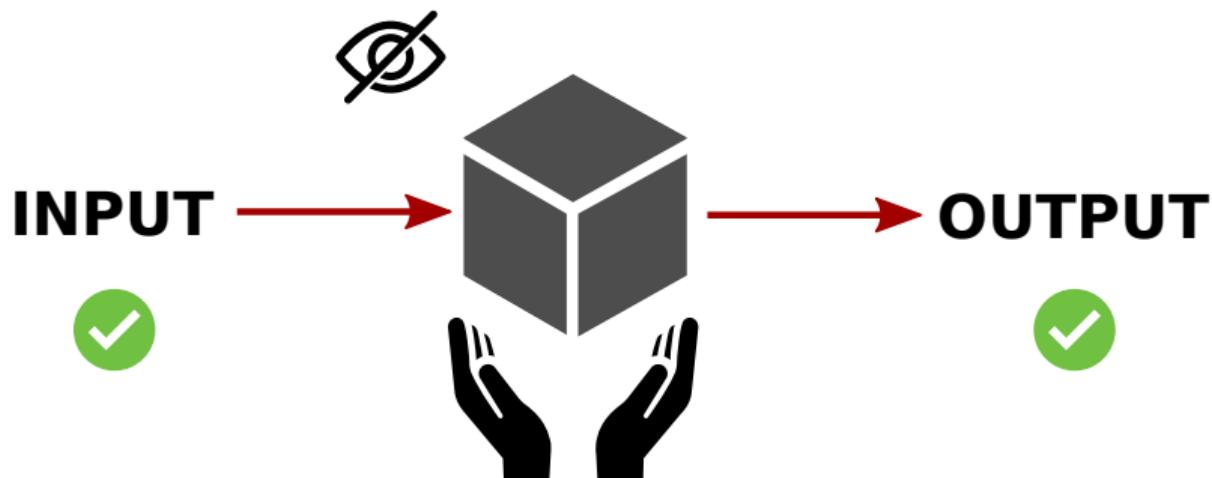
A primer on software security

Side-channels: observe *side-effects* of the computation



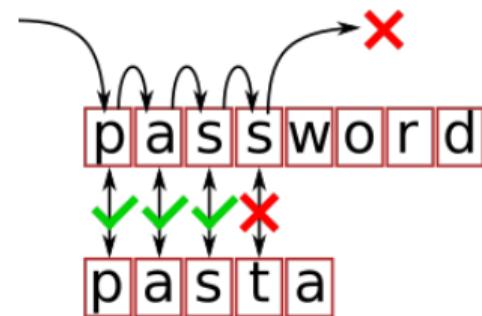
A primer on software security

Constant-time code: eliminate *secret-dependent* side-effects



A vulnerable example program and its constant-time equivalent

```
1 void check_pwd( char *input)
2 {
3     for ( int i=0; i < PWD_LEN; i++)
4         if ( input[ i ] != pwd[ i ])
5             return 0;
6
7     return 1;
8 }
```



Overall execution time reveals correctness of individual password bytes!

→ reduce brute-force attack from an exponential to a linear effort...

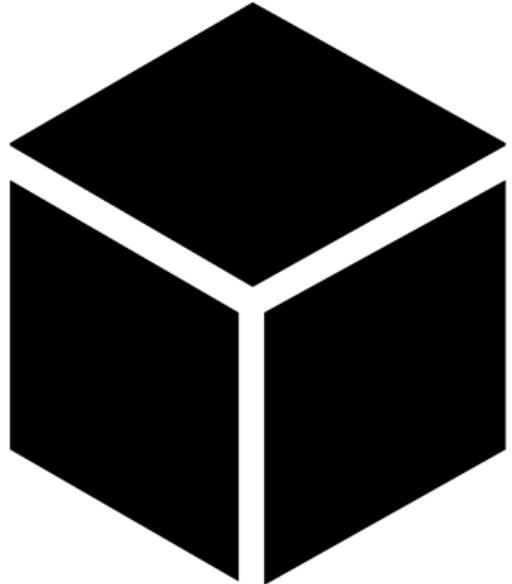
A vulnerable example program and its constant-time equivalent

```
1 void check_pwd(char *input)
2 {
3     for (int i=0; i < PWD_LEN; i++)
4         if (input[i] != pwd[i])
5             return 0;
6
7     return 1;
8 }
```

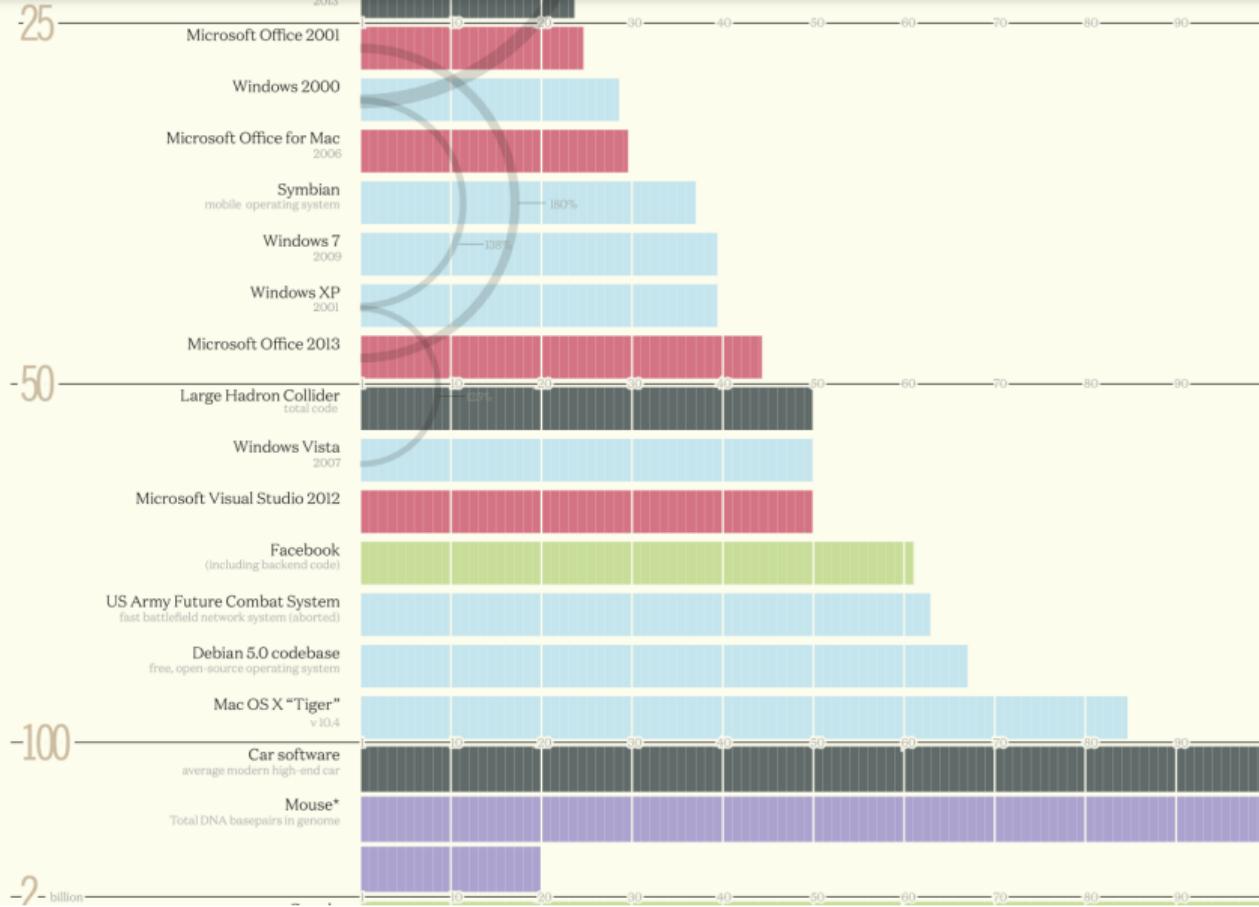
```
1 void check_pwd(char *input)
2 {
3     int rv = 0x0;
4     for (int i=0; i < PWD_LEN; i++)
5         rv |= input[i] ^ pwd[i];
6
7     return (rv == 0);
8 }
```

Rewrite program such that execution time does not depend on secrets

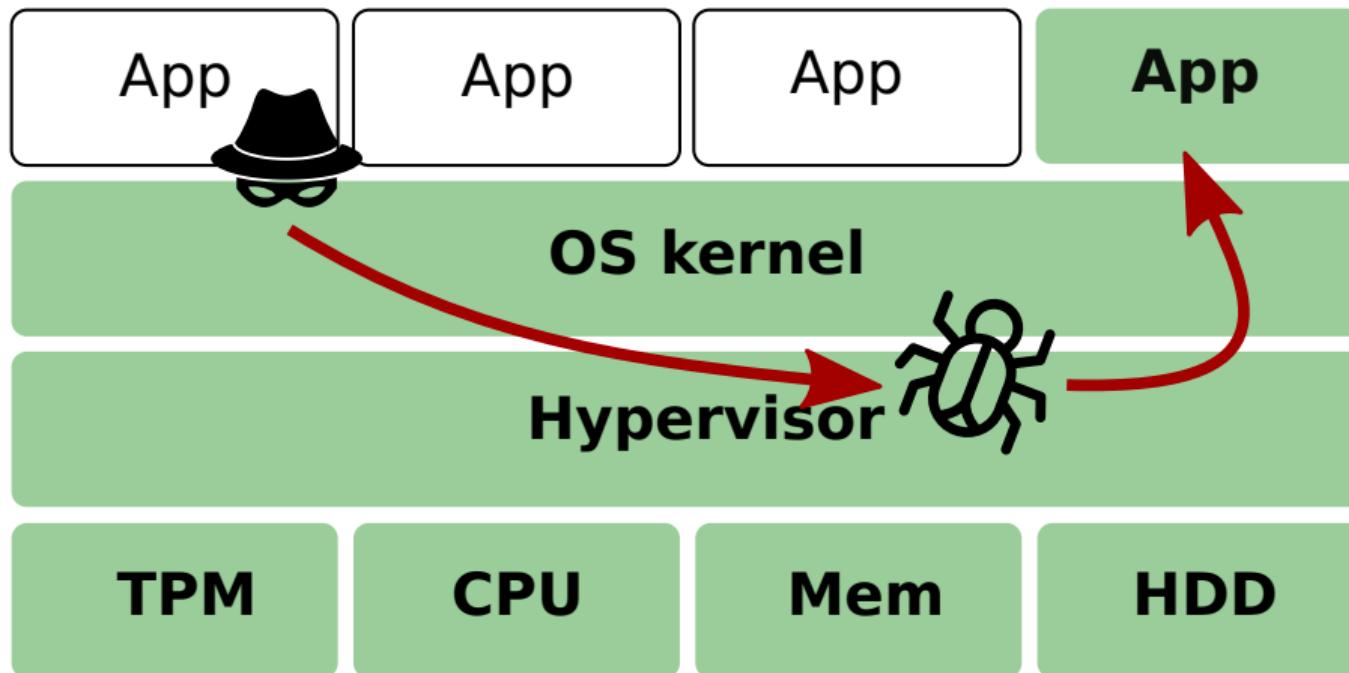
→ manual, error-prone solution; side channels are likely here to stay...



What's inside the black box?

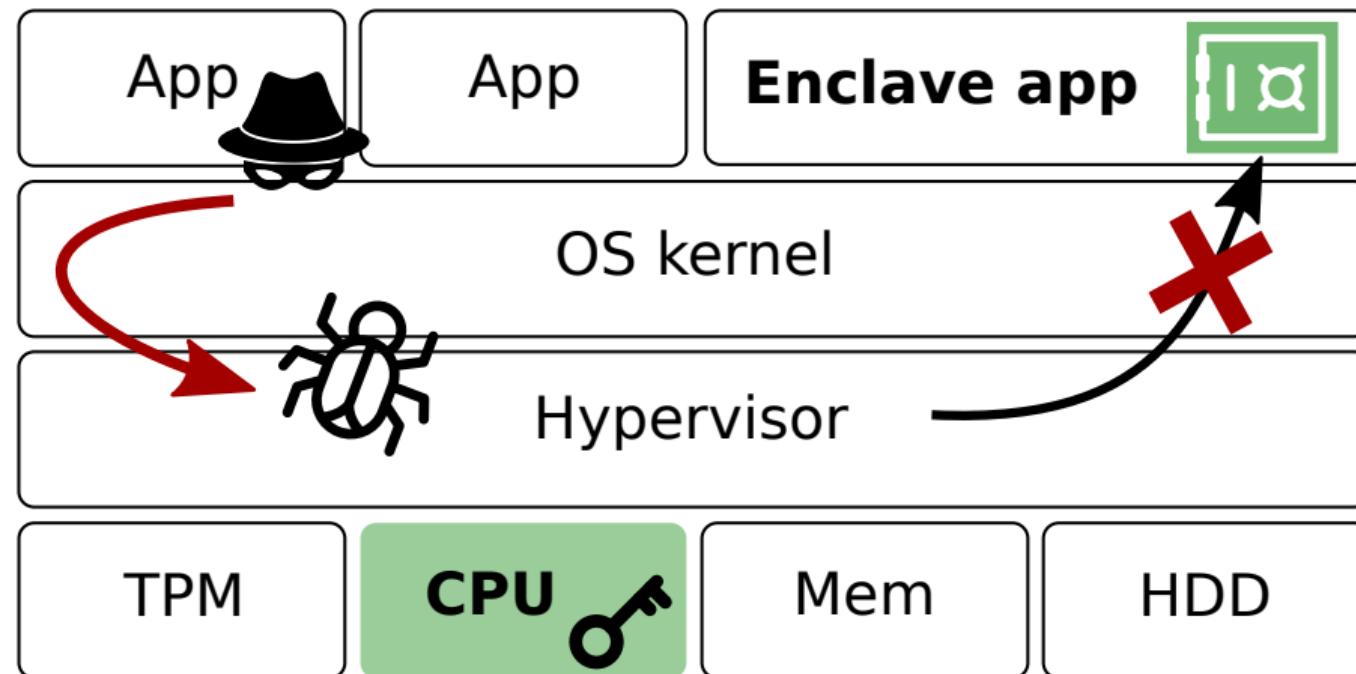


Enclaved execution: Reducing attack surface



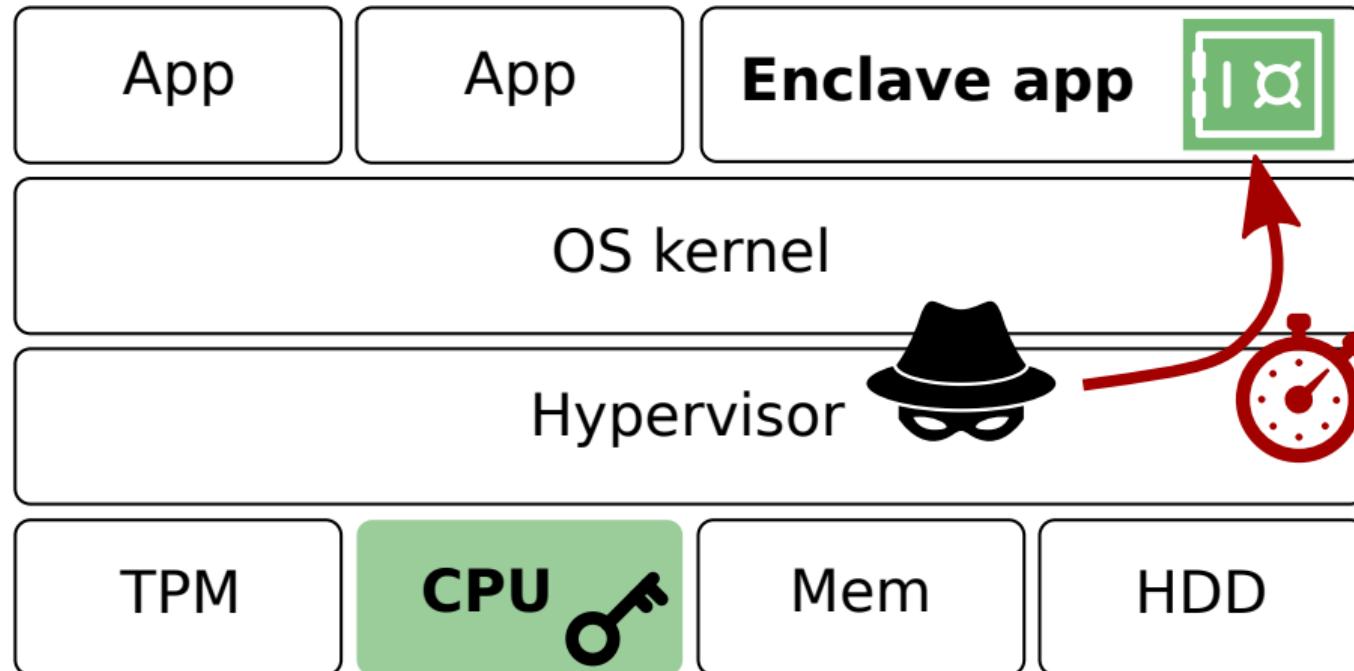
Traditional layered designs: large trusted computing base

Enclaved execution: Reducing attack surface



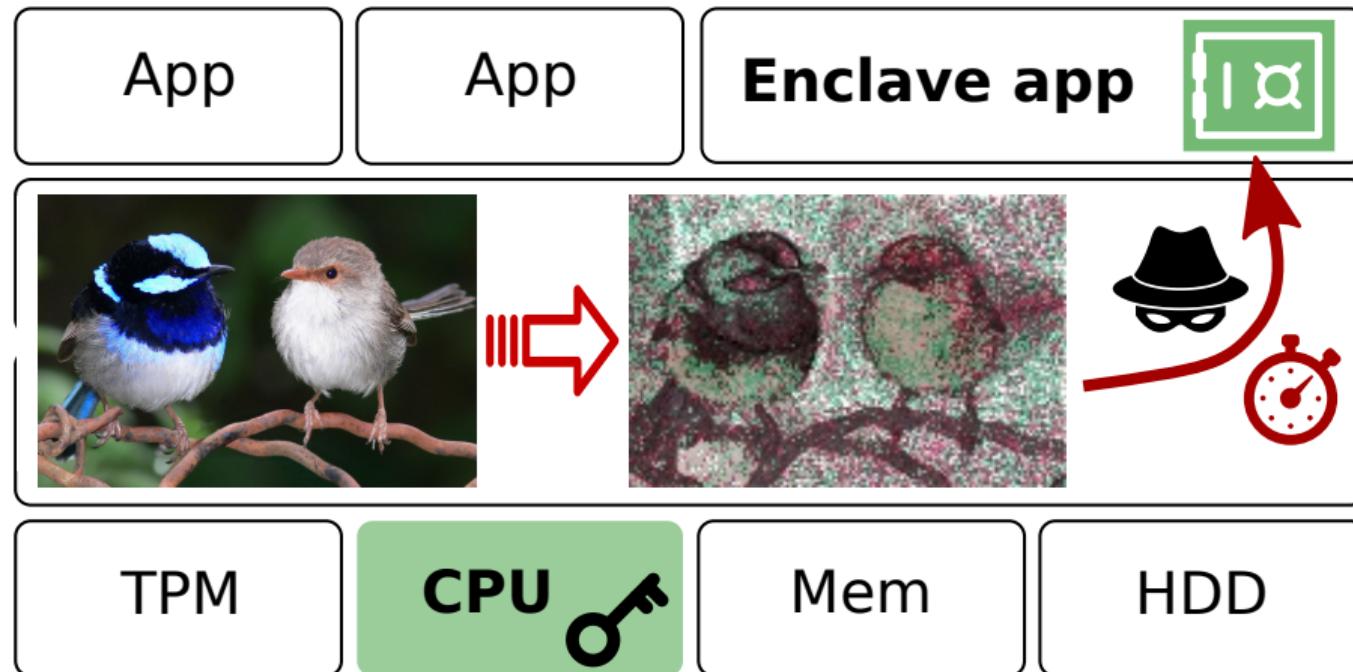
Intel SGX promise: hardware-level **isolation** and **attestation**

Enclaved execution: Privileged side-channel attacks



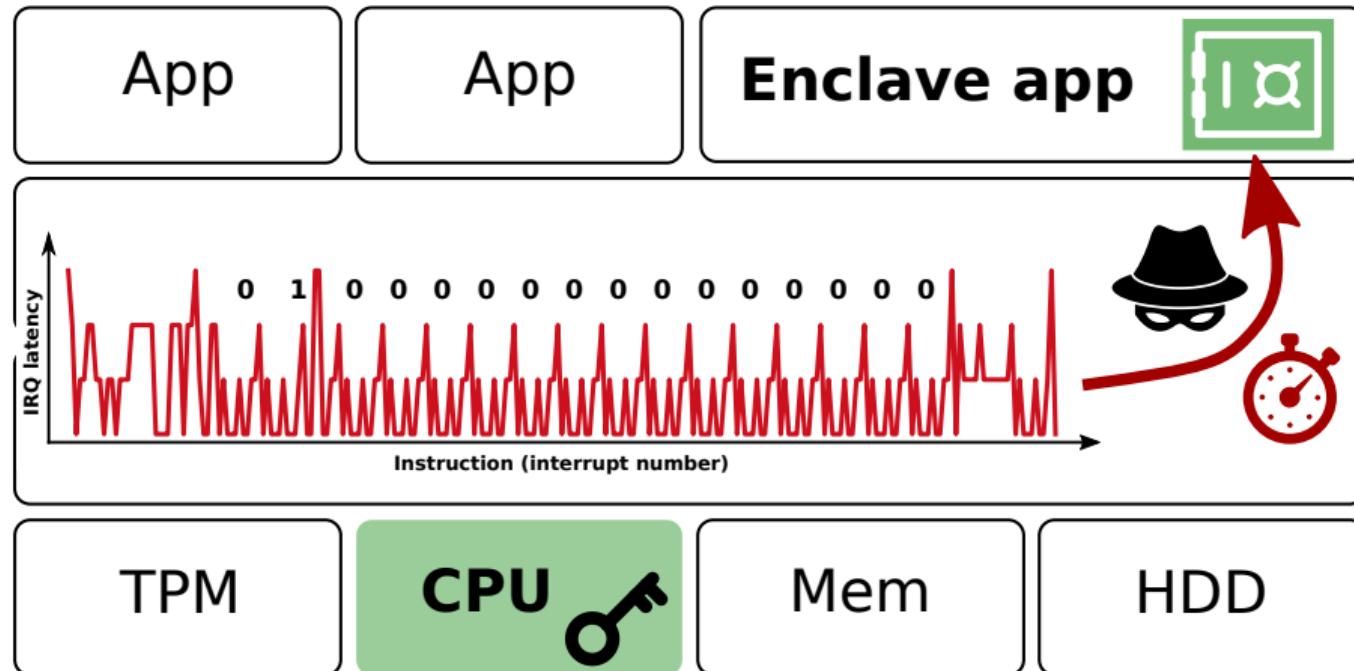
Game-changer: Untrusted OS → new class of powerful **side channels!**

Enclaved execution: Privileged side-channel attacks



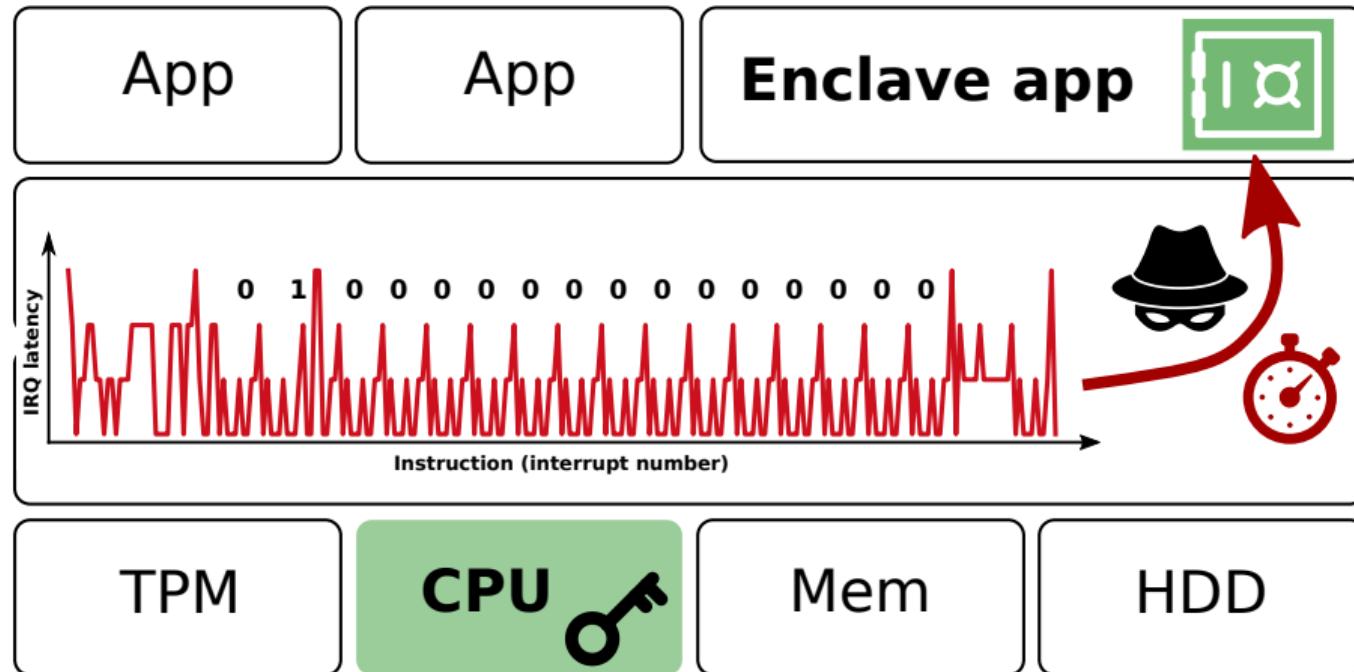
Game-changer: Untrusted OS → new class of powerful **side channels!**

Enclaved execution: Privileged side-channel attacks



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

Enclaved execution: Privileged side-channel attacks



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



KEEP CALM
IT IS
OUT OF SCOPE

Protection from Side-Channel Attacks

Intel® SGX does not provide explicit protection from side-channel attacks. It is the enclave developer's responsibility to address side-channel attack concerns.

In general, enclave operations that require an OCall, such as thread synchronization, I/O, etc., are exposed to the untrusted domain. If using an OCall would allow an attacker to gain insight into enclave secrets, then there would be a security concern. This scenario would be classified as a side-channel attack, and it would be up to the ISV to design the enclave in a way that prevents the leaking of side-channel information.

An attacker with access to the platform can see what pages are being executed or accessed. This side-channel vulnerability can be mitigated by aligning specific code and data blocks to exist entirely within a single page.

More important, the application enclave should use an appropriate crypto implementation that is side channel attack resistant inside the enclave if side-channel attacks are a concern.

Why doesn't Intel eliminate side-channel analysis methods?



Simply put, a side-channel is some observable aspect of a computer system's physical operation, such as timing, power consumption or even sound. As such, they can't be eliminated. However, Intel is committed to rapidly addressing issues such as these as they arise, and providing recommendations through security advisories and security notices. The latest security information on Intel® products can be found [here](#).



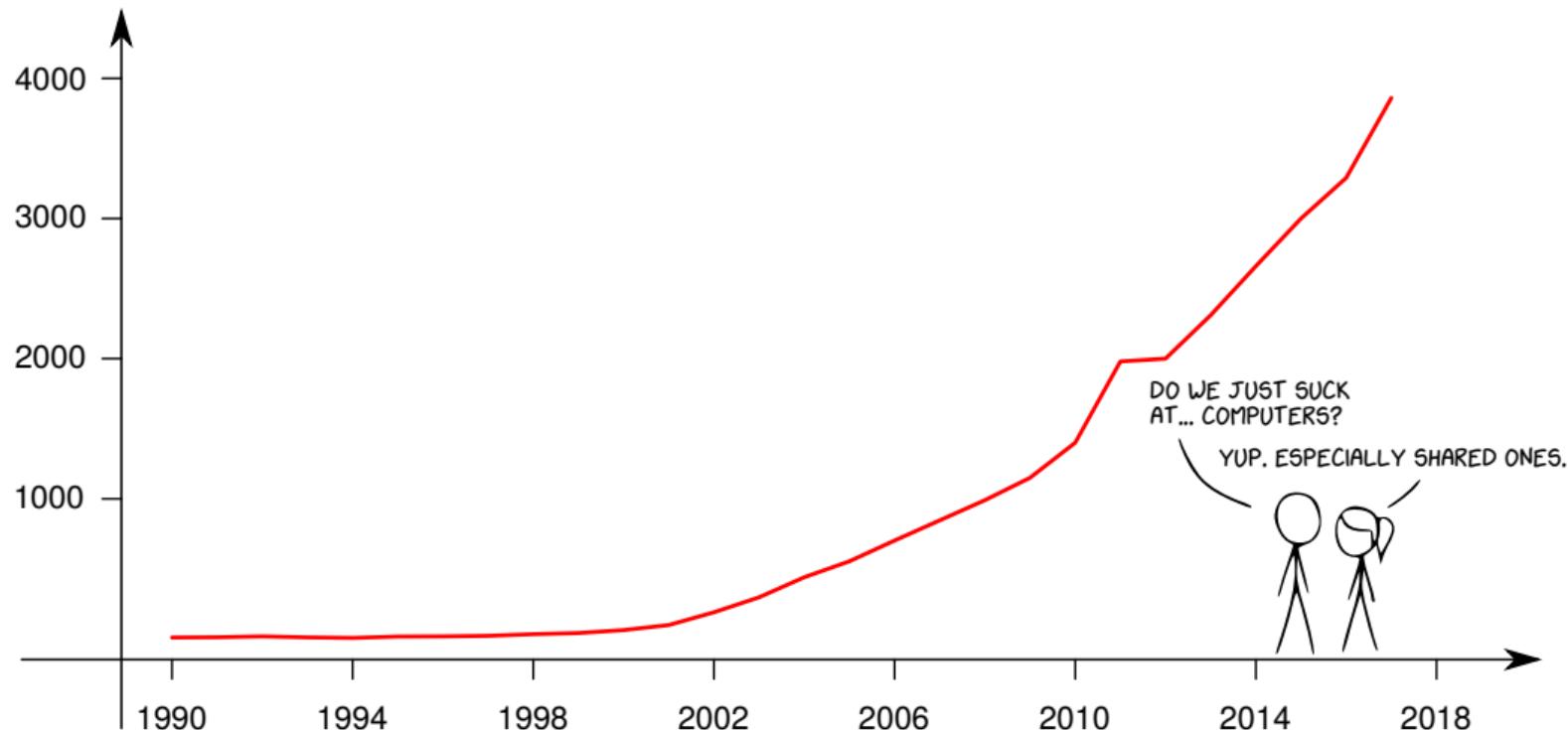
Why doesn't Intel eliminate side-channel analysis methods? ^

Simply put, a side-channel is some observable aspect of a computer system's physical operation, such as timing, power consumption or even sound. As such, they can't be eliminated. However, Intel is committed to rapidly addressing issues such as these as they arise, and providing recommendations through security advisories and security notices. The latest security information on Intel® products can be found [here](#).

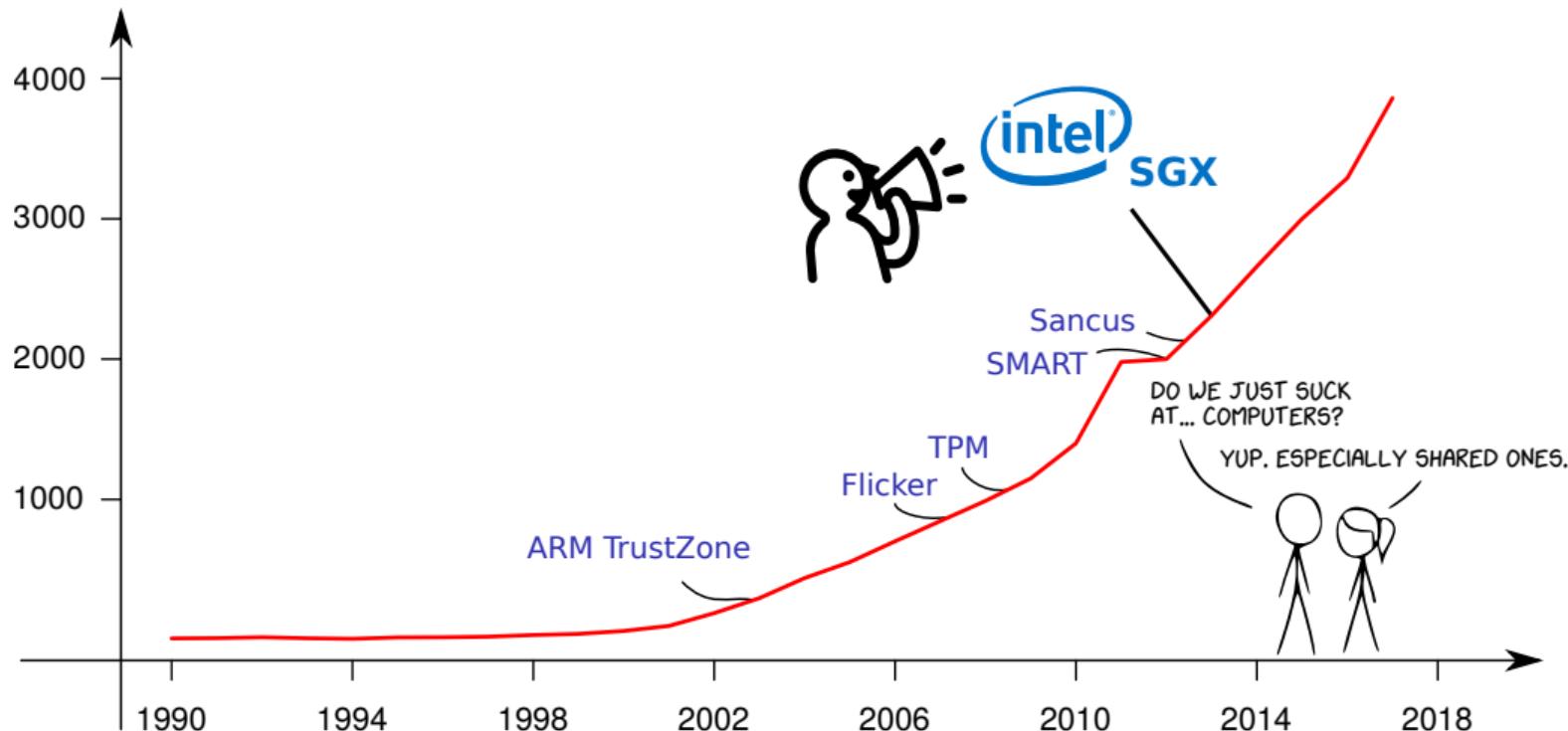


⇒ **Research agenda: systematic understanding of side-channel leakage in TEEs**

Evolution of “side-channel attack” occurrences in Google Scholar

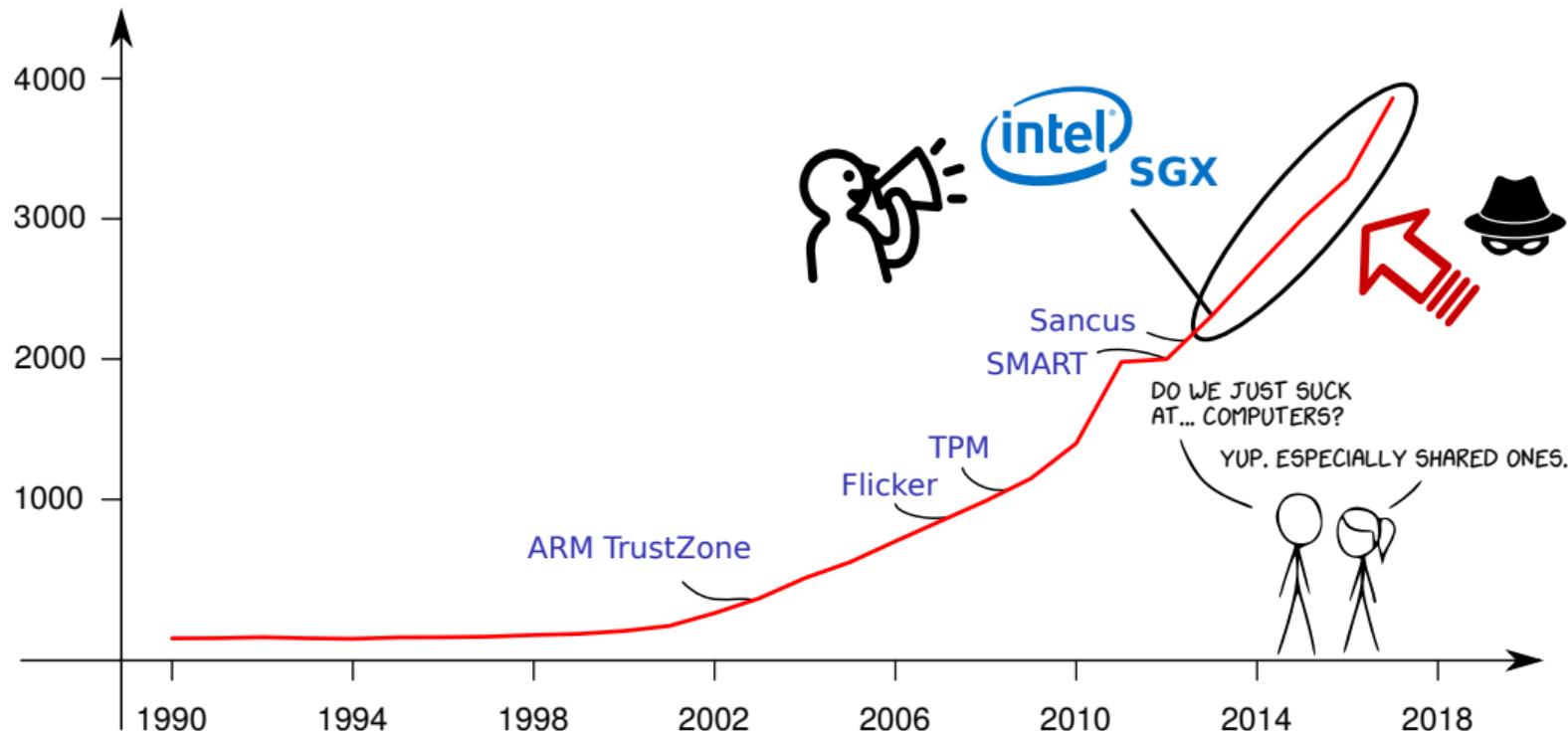


Side-channel attacks and trusted computing



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

Side-channel attacks and trusted computing (focus of today)



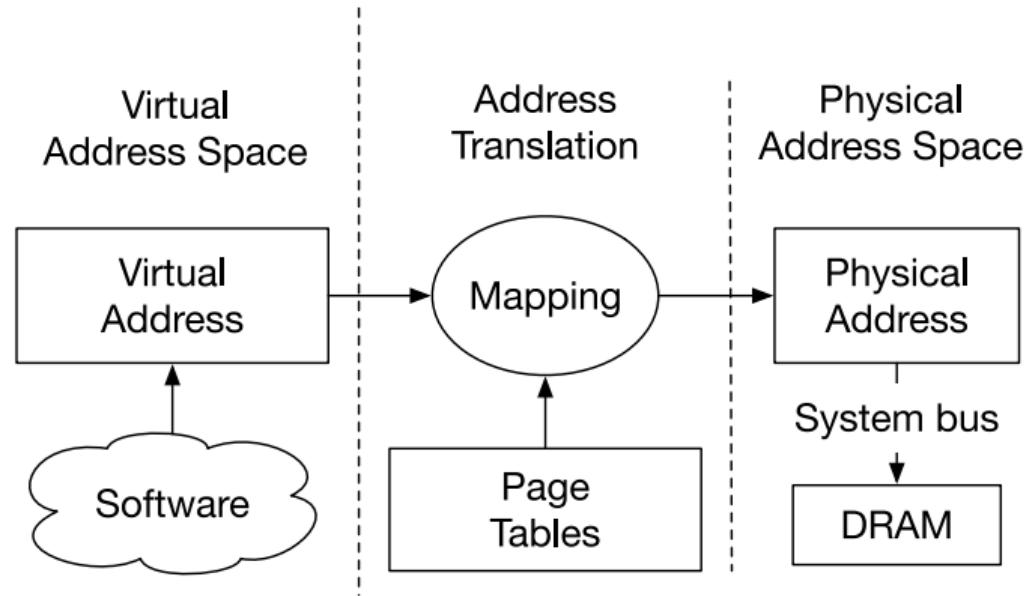
Based on github.com/Pold87/academic-keyword-occurrence and xkcd.com/1938/

Privileged adversary idea #1



Page tables as a side channel?

The virtual memory abstraction



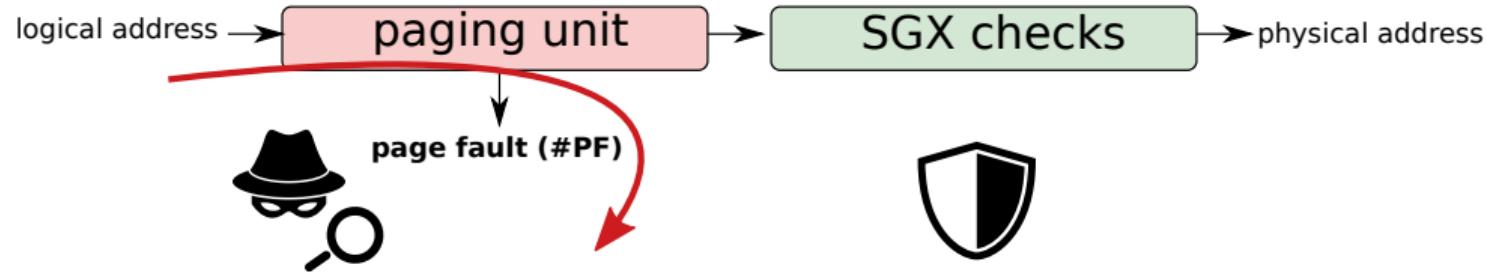
Costan et al. "Intel SGX explained", IACR 2016

Page faults as a side channel



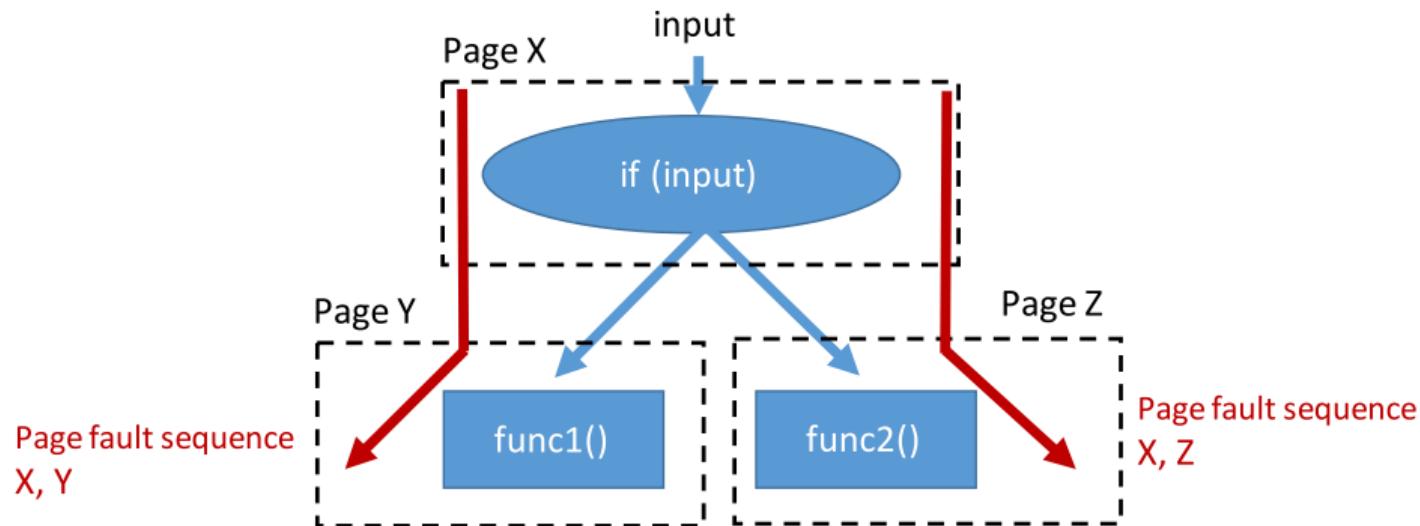
- **SGX machinery** protects against direct address remapping attacks

Page faults as a side channel



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

Page faults as a side channel



- **SGX machinery** protects against direct address remapping attacks
- ... but untrusted address translation may **fault** during enclaved execution (!)
- ⇒ Page fault traces leak **private control/data flow**

#PF attacks: An end-to-end example

```
void inc_secret( void )
{
    if (secret)
        *a += 1;
    else
        *b += 1;
}
```

Page Table

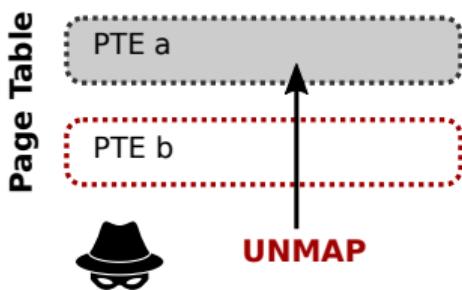
PTE a

PTE b

#PF attacks: An end-to-end example

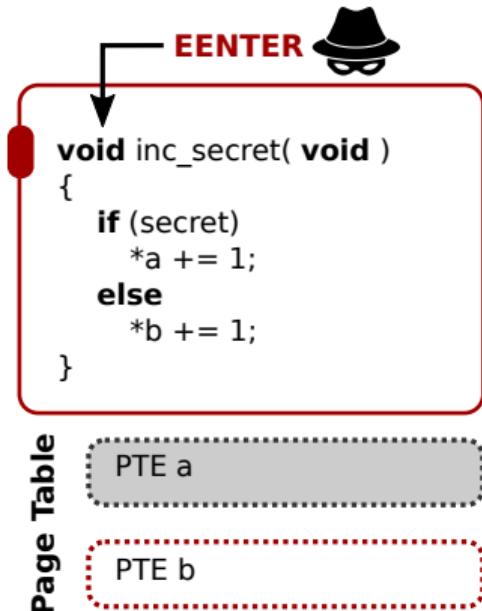
1. Revoke access rights on *unprotected enclave page table entry*

```
void inc_secret( void )  
{  
    if (secret)  
        *a += 1;  
    else  
        *b += 1;  
}
```



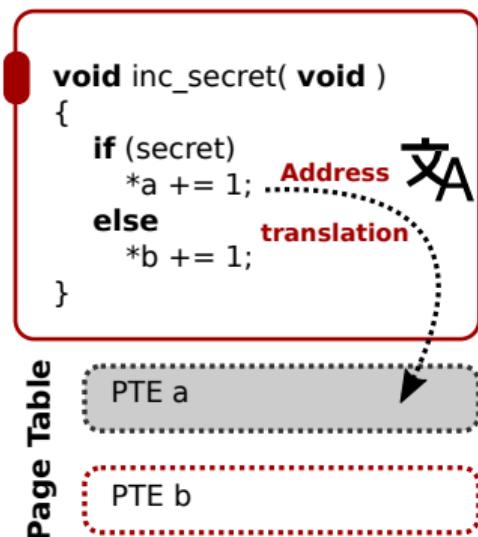
#PF attacks: An end-to-end example

1. Revoke access rights on *unprotected enclave page table entry*
2. **Enter** victim enclave



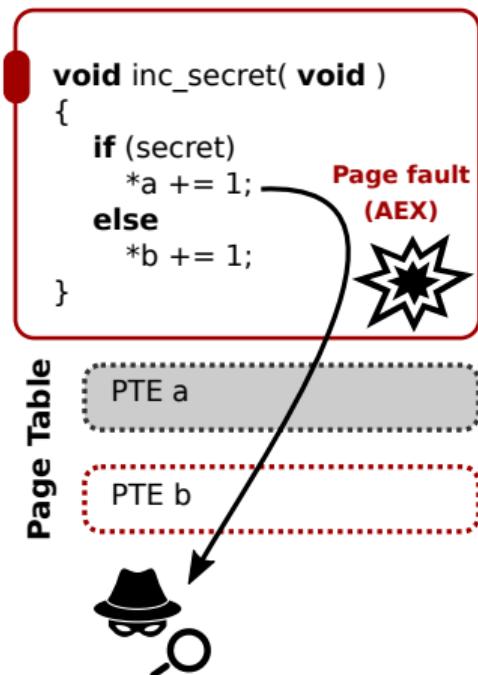
#PF attacks: An end-to-end example

1. Revoke access rights on *unprotected enclave page table entry*
2. Enter victim enclave
3. Secret-dependent **data memory access**
 - ~ Processor reads page table setup by untrusted OS



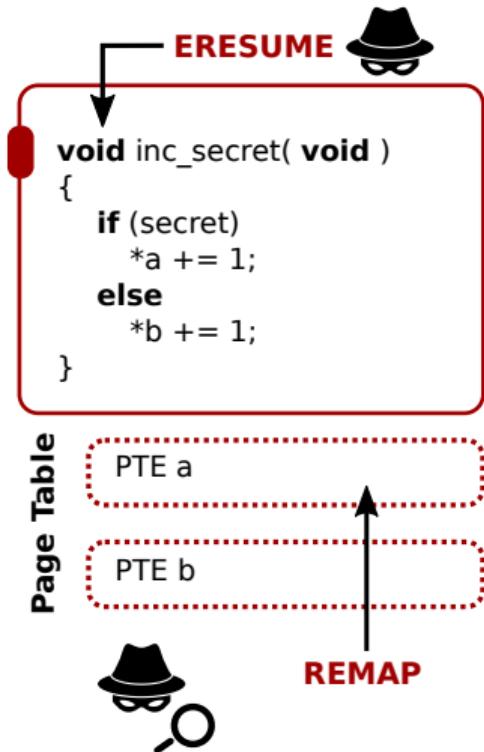
#PF attacks: An end-to-end example

1. Revoke access rights on *unprotected enclave page table entry*
2. Enter victim enclave
3. Secret-dependent **data memory access**
 - ~ Processor reads page table setup by *untrusted OS*
4. Virtual address not present → raise **page fault**
 - ~ Processor exits enclave and vectors to untrusted OS

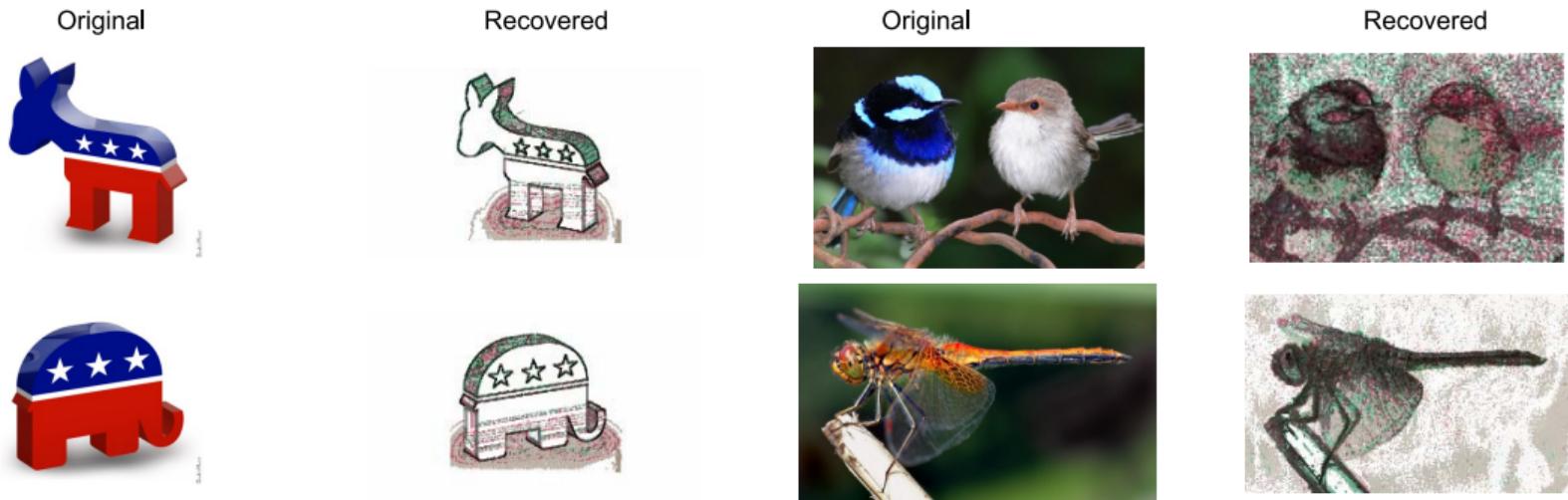


#PF attacks: An end-to-end example

1. Revoke access rights on *unprotected enclave page table entry*
2. Enter victim enclave
3. Secret-dependent *data memory access*
 - ~ Processor reads page table setup by untrusted OS
4. Virtual address not present → raise *page fault*
 - ~ Processor exits enclave and vectors to untrusted OS
5. Restore access rights and *resume* victim enclave



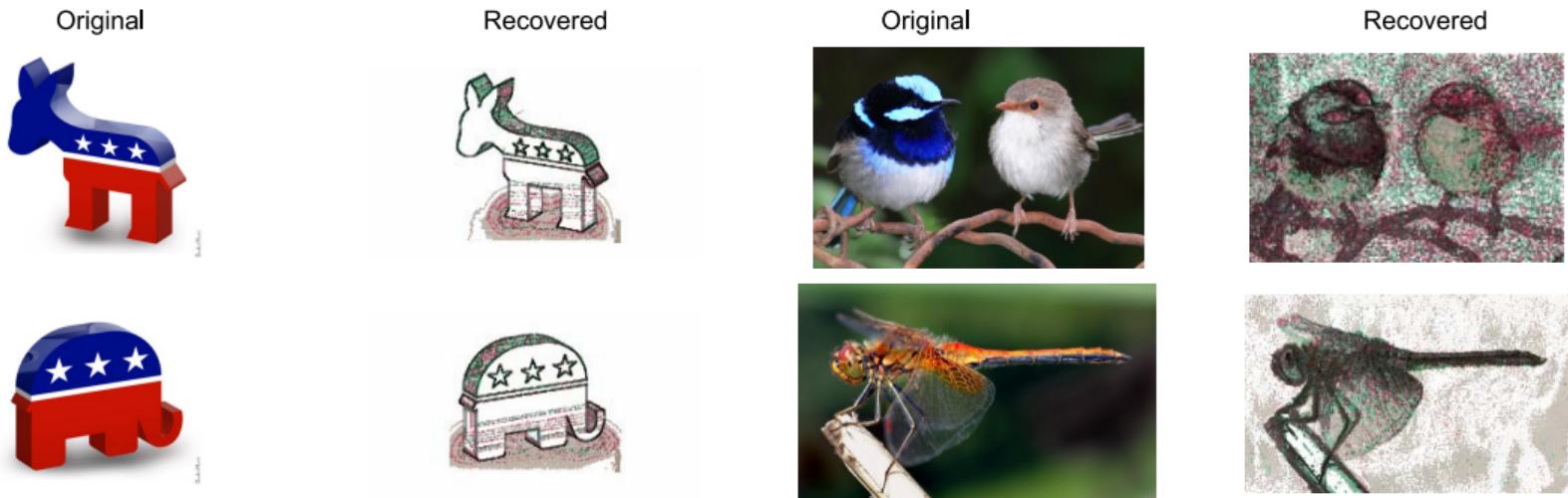
Page table-based attacks in practice



Xu et al.: "Controlled-channel attacks: Deterministic side channels for untrusted operating systems", Oakland 2015

⇒ **Low-noise, single-run** exploitation of legacy applications

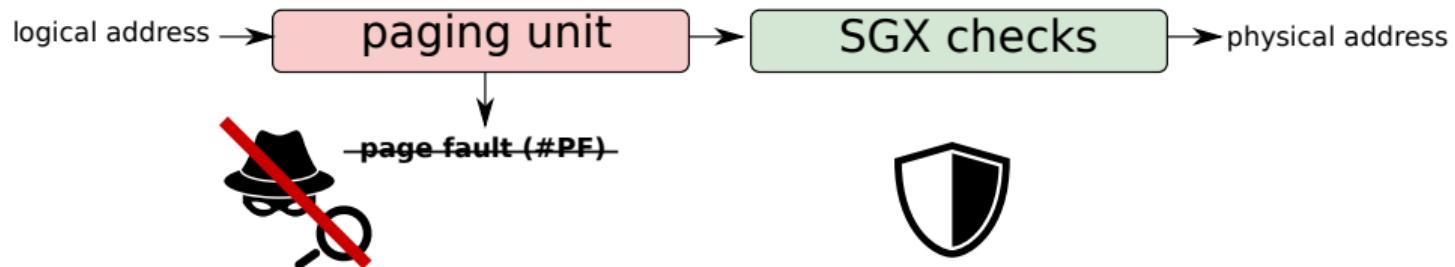
Page table-based attacks in practice



Xu et al.: "Controlled-channel attacks: Deterministic side channels for untrusted operating systems", Oakland 2015

... but at a relative coarse-grained **4 KiB granularity**

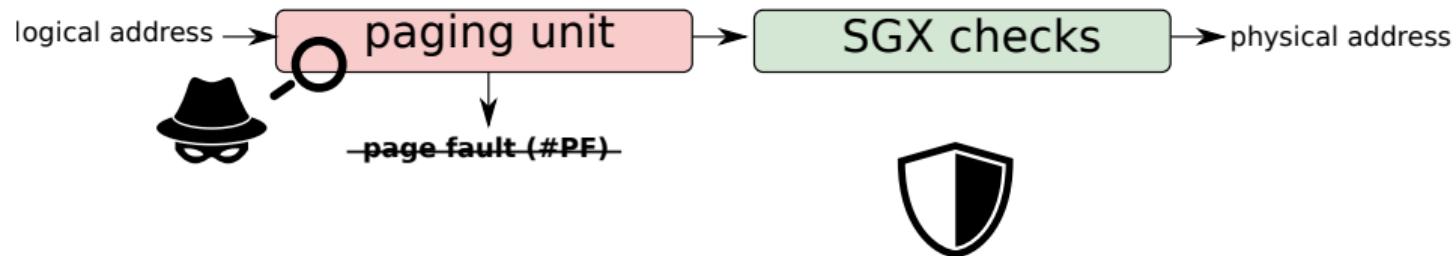
Naive solutions: Hiding enclave page faults



Shih et al. "T-SGX: Eradicating controlled-channel attacks against enclave programs", NDSS 2017

Shinde et al. "Preventing page faults from telling your secrets", AsiaCCS 2016

Naive solutions: Hiding enclave page faults



... But stealthy attacker can still learn page accesses without triggering faults!

Documented side-effects of address translation

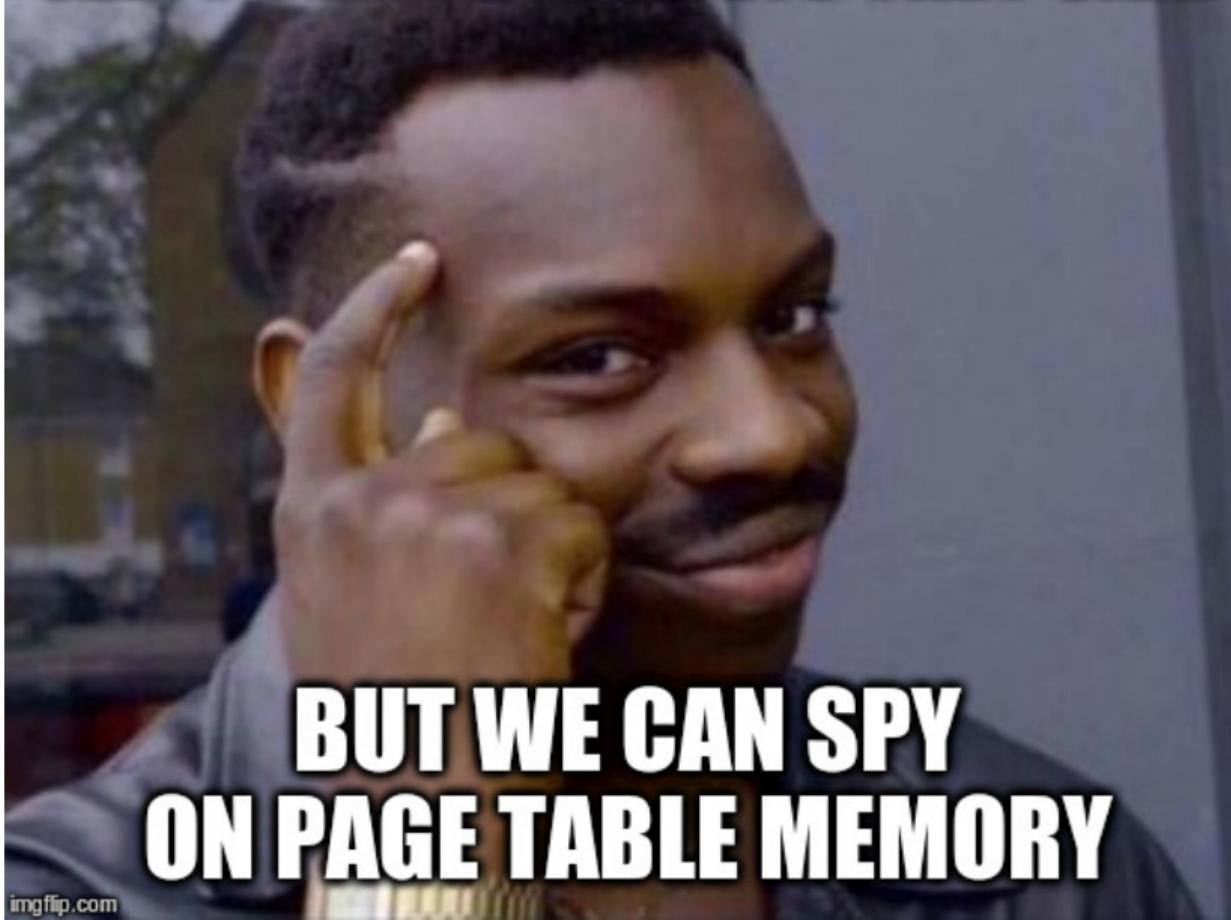
4.8 ACCESSED AND DIRTY FLAGS

For any paging-structure entry that is used during linear-address translation, bit 5 is the **accessed** flag.² For paging-structure entries that map a page (as opposed to referencing another paging structure), bit 6 is the **dirty** flag. These flags are provided for use by memory-management software to manage the transfer of pages and paging structures into and out of physical memory.

Whenever the processor uses a paging-structure entry as part of linear-address translation, it sets the accessed flag in that entry (if it is not already set).

Whenever there is a write to a linear address, the processor sets the dirty flag (if it is not already set) in the paging-structure entry that identifies the final physical address for the linear address (either a PTE or a paging-structure entry in which the PS flag is 1).

CAN'T SEE PAGE FAULTS THEY SAID



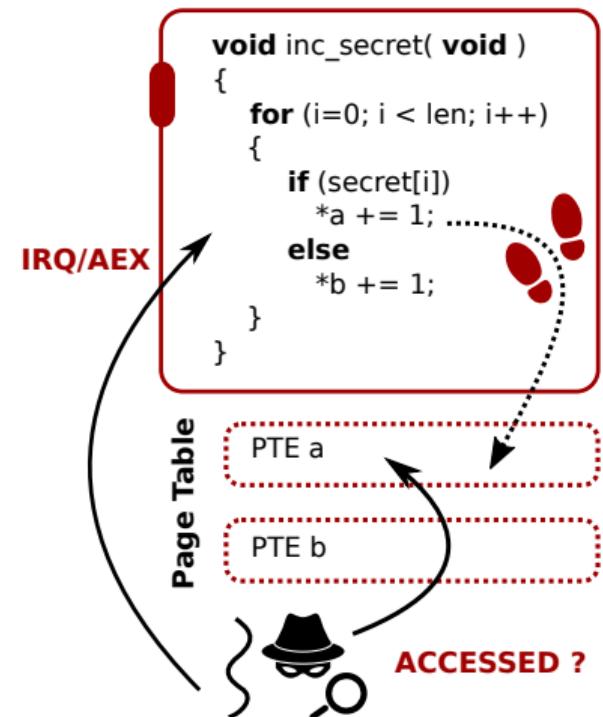
BUT WE CAN SPY
ON PAGE TABLE MEMORY

Telling your secrets without page faults

1. Attack vector: PTE status flags:

- A(ccessed) bit
- D(irty) bit

~ Also updated in enclave mode!



Telling your secrets without page faults

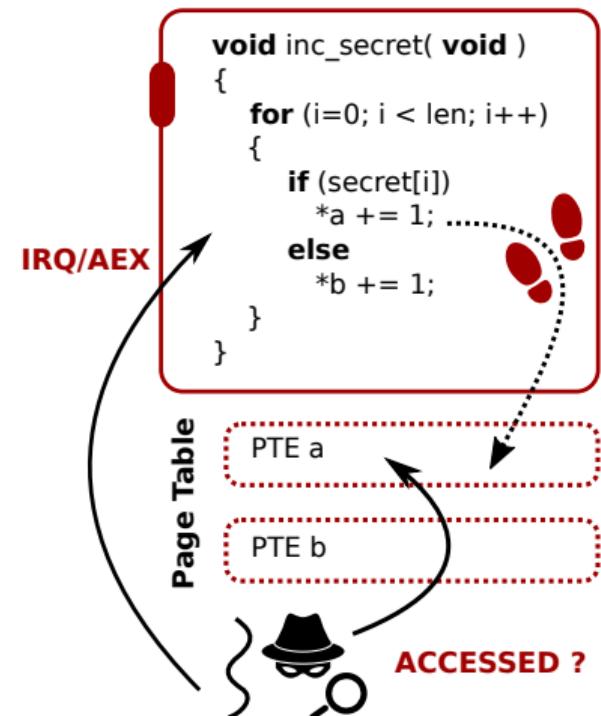
1. Attack vector: PTE status flags:

- A(ccessed) bit
- D(irty) bit

~ Also updated in enclave mode!

2. Attack vector: Unprotected page table memory:

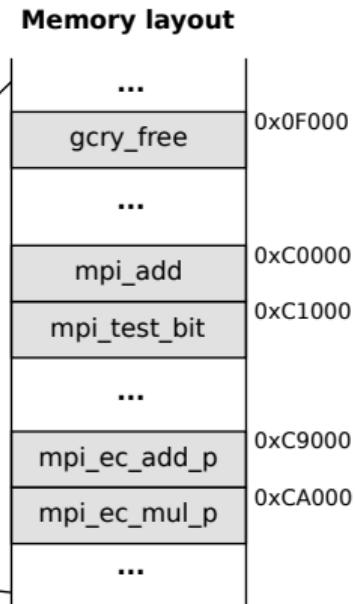
- Cached as regular data
 - Accessed during address translation
- ~ Flush+Reload cache timing attack!



Attacking Libgcrypt EdDSA (simplified)

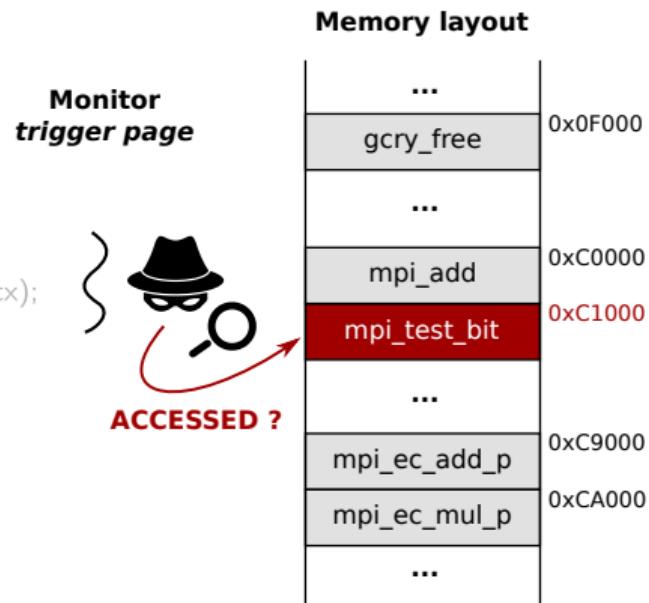
```
1 if (mpi_is_secure (scalar)) {
2     /* If SCALAR is in secure memory we assume that it is the
3         secret key we use constant time operation. */
4     point_init (&tmppt);
5
6     for (j=nbits-1; j >= 0; j--) {
7         -gcry_mpi_ec_dup_point (result, result, ctx);
8         -gcry_mpi_ec_add_points (&tmppt, result, point, ctx);
9         point_swap_cond (result, &tmppt, mpi_test_bit (scalar, j), ctx);
10    }
11    point_free (&tmppt);
12 } else {
13     for (j=nbits-1; j >= 0; j--) {
14         -gcry_mpi_ec_dup_point (result, result, ctx);
15         if (mpi_test_bit (scalar, j))
16             -gcry_mpi_ec_add_points (result, result, point, ctx);
17     }
18 }
```

22 Code pages per iteration



Attacking Libgcrypt EdDSA (simplified)

```
1 if (mpi_is_secure (scalar)) {
2     /* If SCALAR is in secure memory we assume that it is the
3         secret key we use constant time operation. */
4     point_init (&tmppnt);
5
6     for (j=nbits-1; j >= 0; j--) {
7         -gcry_mpi_ec_dup_point (result, result, ctx);
8         -gcry_mpi_ec_add_points (&tmppnt, result, point, ctx);
9         point_swap_cond (result, &tmppnt, mpi_test_bit (scalar, j), ctx);
10    }
11    point_free (&tmppnt);
12 } else {
13     for (j=nbits-1; j >= 0; j--) {
14         -gcry_mpi_ec_dup_point (result, result, ctx);
15         if (mpi_test_bit (scalar, j))
16             -gcry_mpi_ec_add_points (result, result, point, ctx);
17     }
18 }
```



Attacking Libgcrypt EdDSA (simplified)

```
1 if (mpi_is_secure (scalar)) {
2     /* If SCALAR is in secure memory we assume that it is the
3         secret key we use constant time operation. */
4     point_init (&tmppnt);
5
6     for (j=nbits-1; j >= 0; j--) {
7         -gcry_mpi_ec_dup_point (result, result, ctx);
8         -gcry_mpi_ec_add_points (&tmppnt, result, point, ctx);
9         point_swap_cond (result, &tmppnt, mpi_test_bit (scalar, j), ctx);
10    }
11    point_free (&tmppnt);
12 } else {
13     for (j=nbits-1; j >= 0; j--) {
14         -gcry_mpi_ec_dup_point (result, result, ctx);
15         if (mpi_test_bit (scalar, j))
16             -gcry_mpi_ec_add_points (result, result, point, ctx);
17    }
18 }
```

INTERRUPT

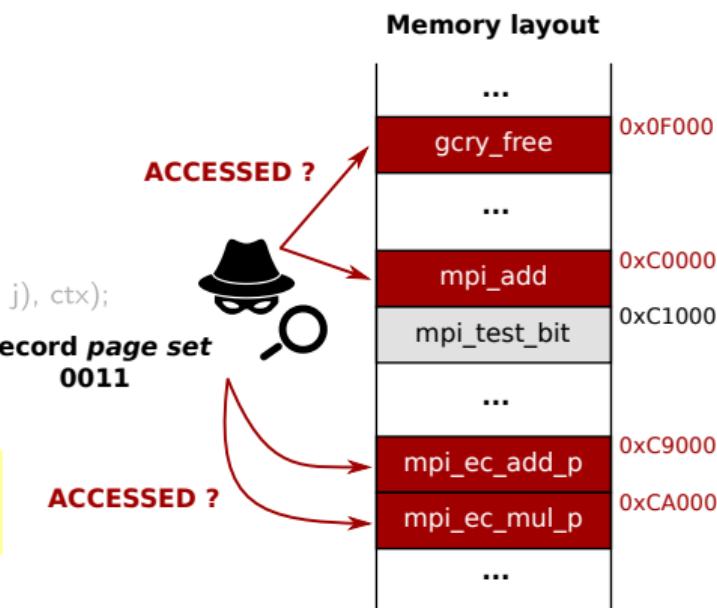


Memory layout

...	0x0F000
gcry_free	
...	0xC0000
mpi_add	0xC1000
mpi_test_bit	
...	0xC9000
mpi_ec_add_p	
mpi_ec_mul_p	0xCA000
...	

Attacking Libgcrypt EdDSA (simplified)

```
1 if (mpi_is_secure (scalar)) {
2     /* If SCALAR is in secure memory we assume that it is the
3         secret key we use constant time operation. */
4     point_init (&tmppt);
5
6     for (j=nbits-1; j >= 0; j--) {
7         -gcry_mpi_ec_dup_point (result, result, ctx);
8         -gcry_mpi_ec_add_points (&tmppt, result, point, ctx);
9         point_swap_cond (result, &tmppt, mpi_test_bit (scalar, j), ctx);
10    }
11    point_free (&tmppt);
12 } else {
13     for (j=nbits-1; j >= 0; j--) {
14         -gcry_mpi_ec_dup_point (result, result, ctx);
15         if (mpi_test_bit (scalar, j))
16             -gcry_mpi_ec_add_points (result, result, point, ctx);
17     }
18 }
```



Attacking Libgcrypt EdDSA (simplified)

```
1 if (mpi_is_secure (scalar)) {
2     /* If SCALAR is in secure memory we assume that it is the
3         secret key we use constant time operation. */
4     point_init (&tmppt);
5
6     for (j=nbits-1; j >= 0; j--) {
7         -gcry_mpi_ec_dup_point (result, result, ctx);
8         -gcry_mpi_ec_add_points (&tmppt, result, point, ctx);
9         point_swap_cond (result, &tmppt, mpi_test_bit (scalar, j), ctx);
10    }
11    point_free (&tmppt);
12 } else {
13     for (j=nbits-1; j >= 0; j--) {
14         -gcry_mpi_ec_dup_point (result, result, ctx);
15         if (mpi_test_bit (scalar, j))
16             -gcry_mpi_ec_add_points (result, result, point, ctx);
17    }
18 }
```

Full 512-bit key recovery, single run



Memory layout	
...	0x0F000
gcry_free	0xC0000
...	0xC1000
mpi_add	0xC9000
mpi_test_bit	0xCA000
...	

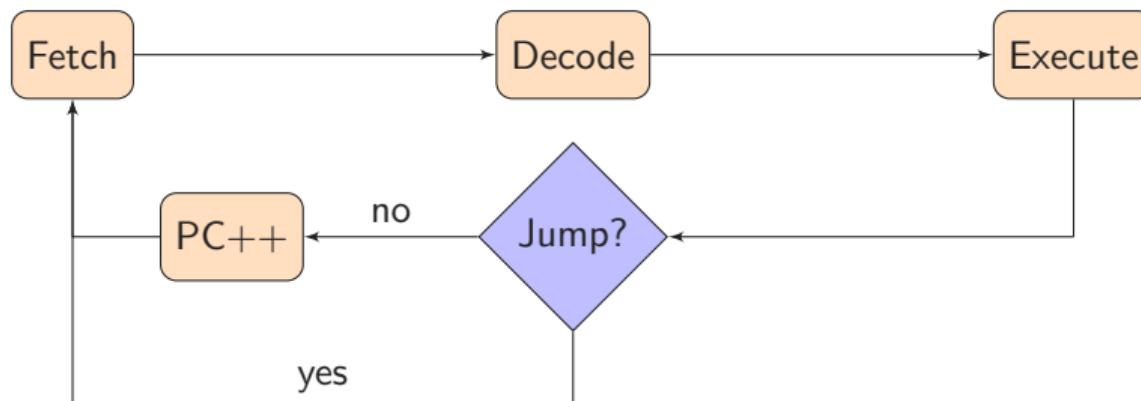
Privileged adversary idea #2



Interrupts as a side channel?

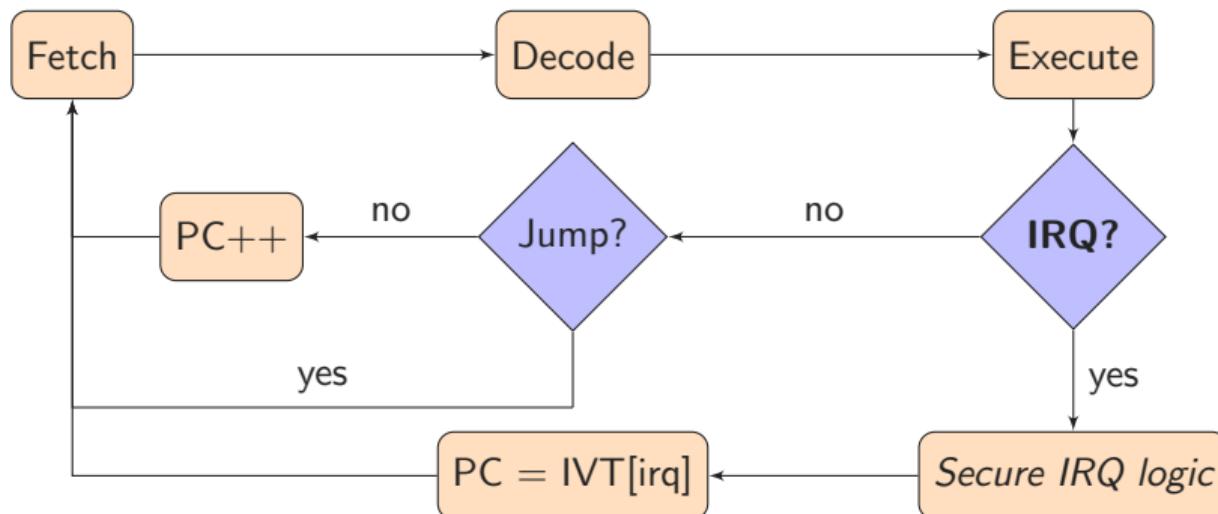
Back to basics: Fetch-decode-execute

Elementary CPU behavior: stored program computer



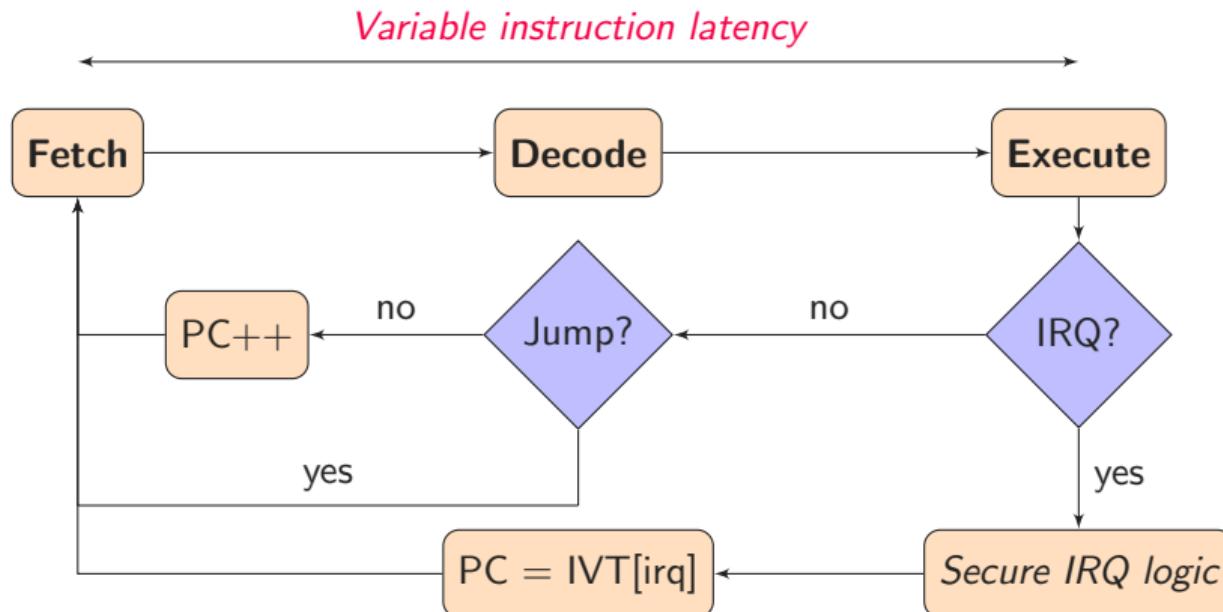
Back to basics: Fetch-decode-execute

Interrupts: asynchronous real-world events, handled on instruction retirement

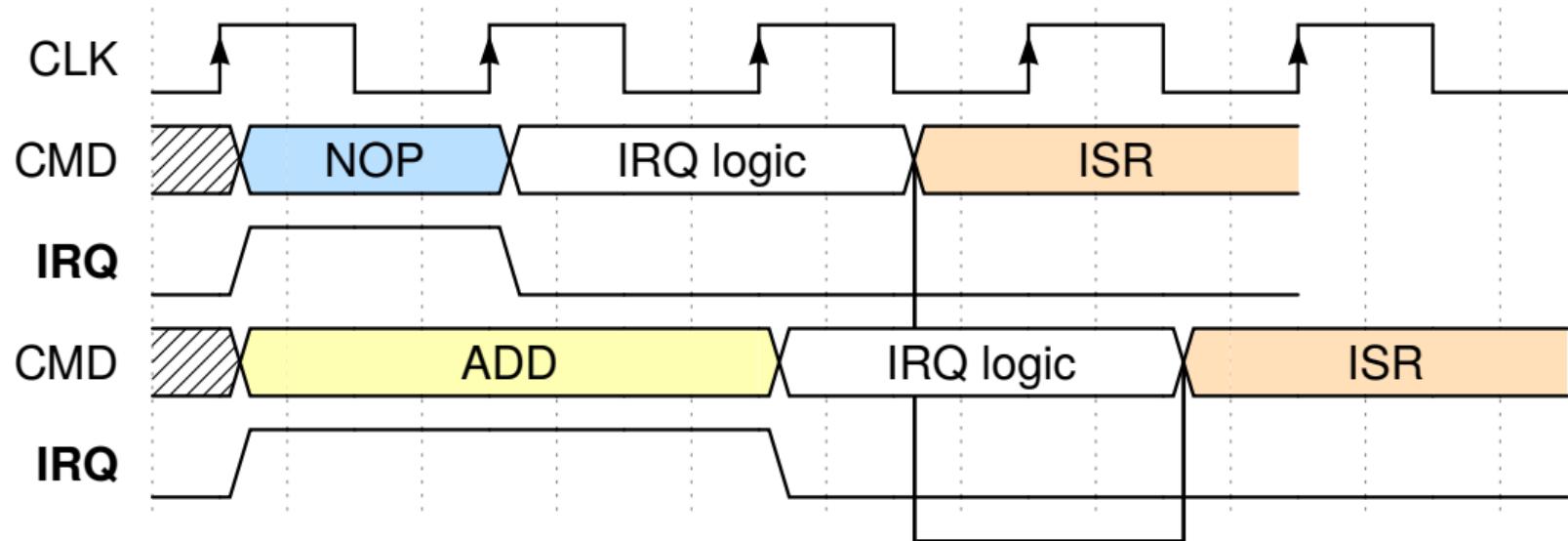


Back to basics: Fetch-decode-execute

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



Wait a cycle: Interrupt latency as a side channel



```
if (secret){ ADD @R5+, R6; } // 2 cycles
else        { NOP; NOP; } // 2*1 cycle
```



Attacking a Sancus application with interrupt latency



Attacking a Sancus application with interrupt latency

Driver enclave: 16-bit vector indicates which keys are down

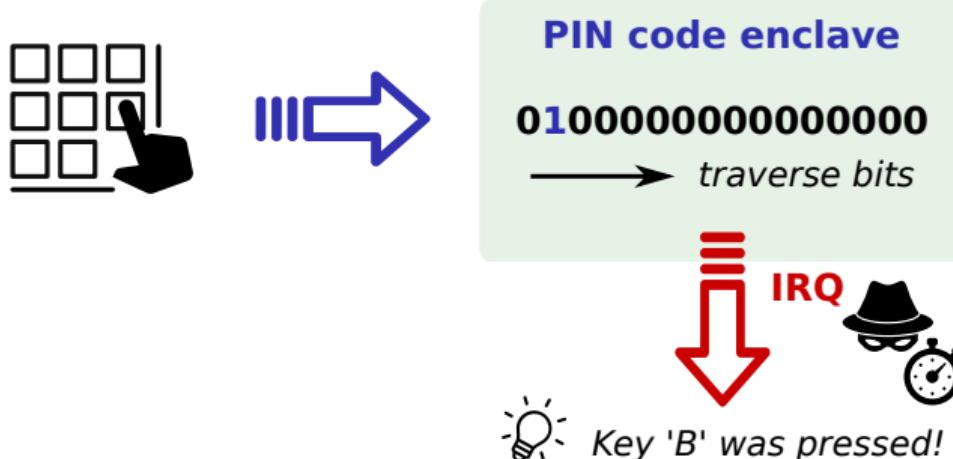


PIN code enclave

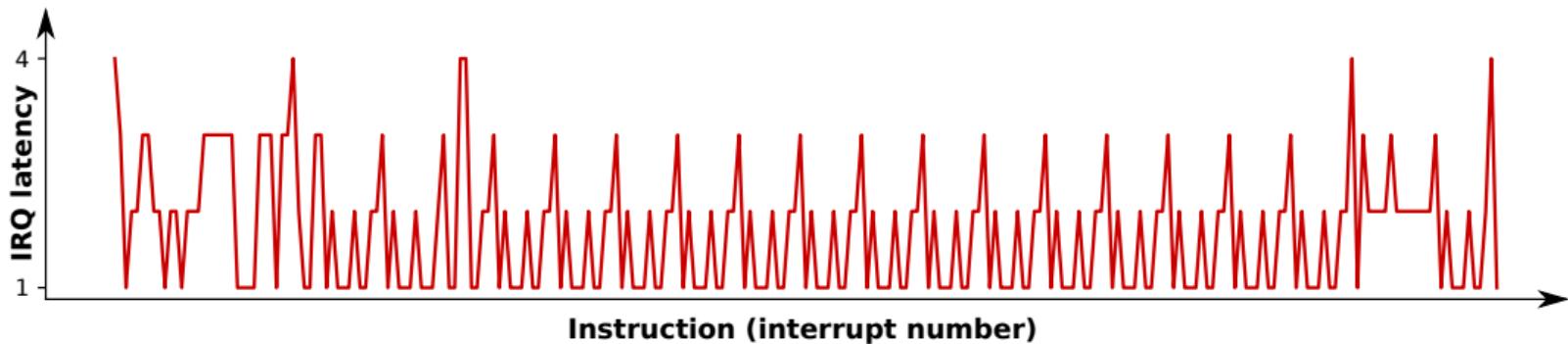
0100000000000000
→ *traverse bits*

Attacking a Sancus application with interrupt latency

Attacker: Interrupt *conditional control flow* to infer secret PIN

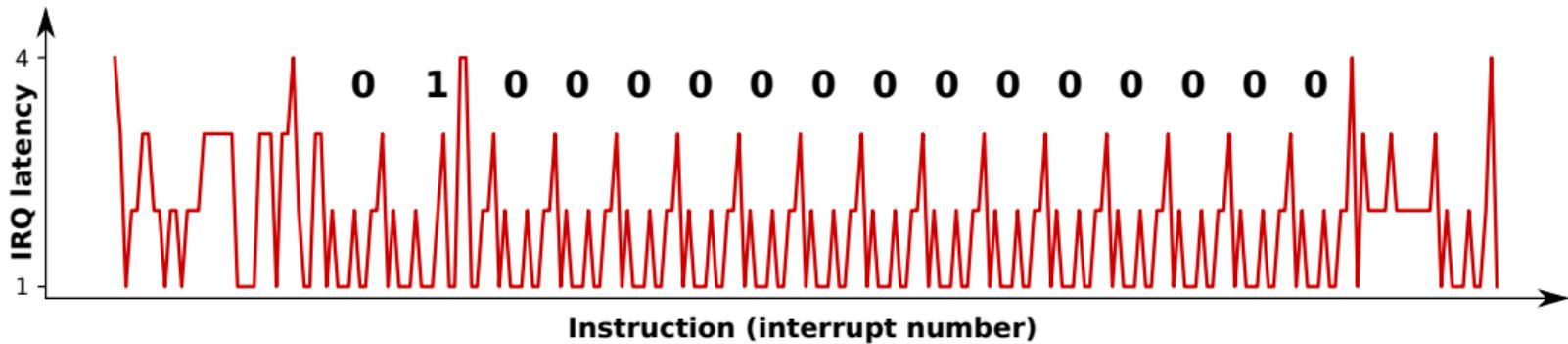


Sancus IRQ timing attack: Inferring key strokes



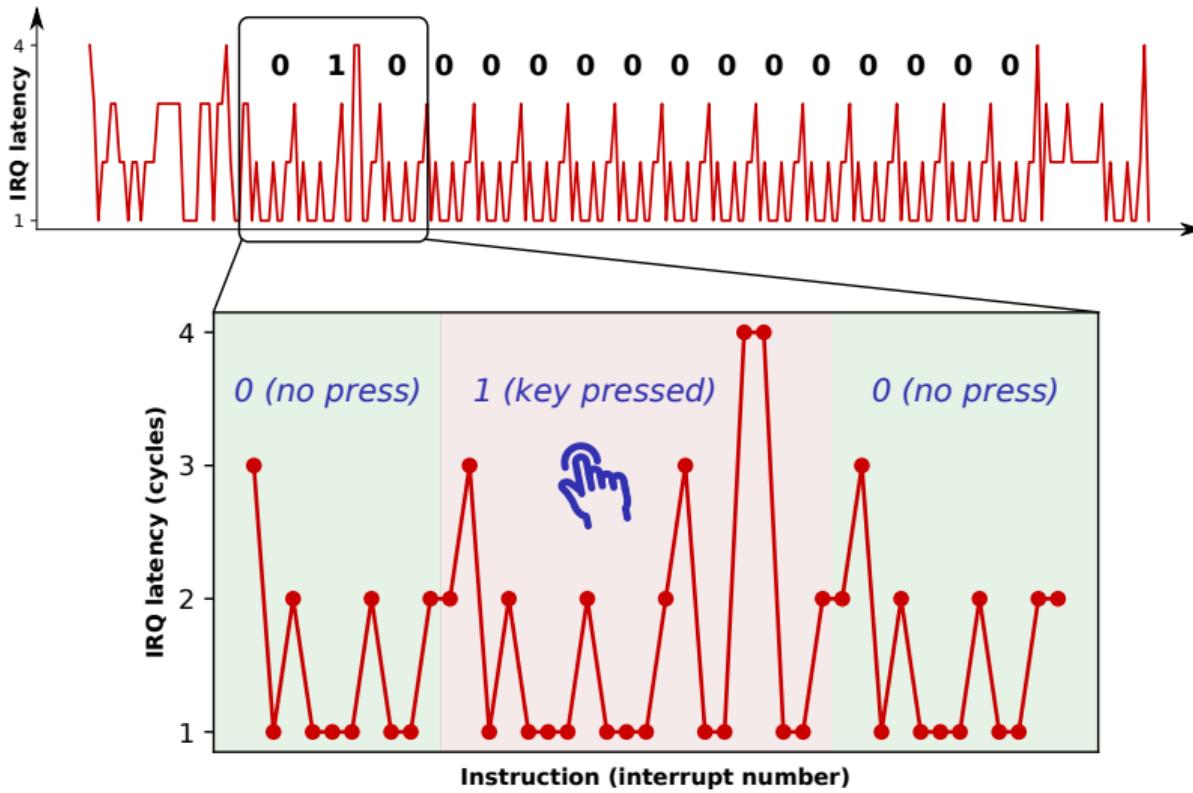
Enclave x-ray: Start-to-end trace enclaved execution

Sancus IRQ timing attack: Inferring key strokes

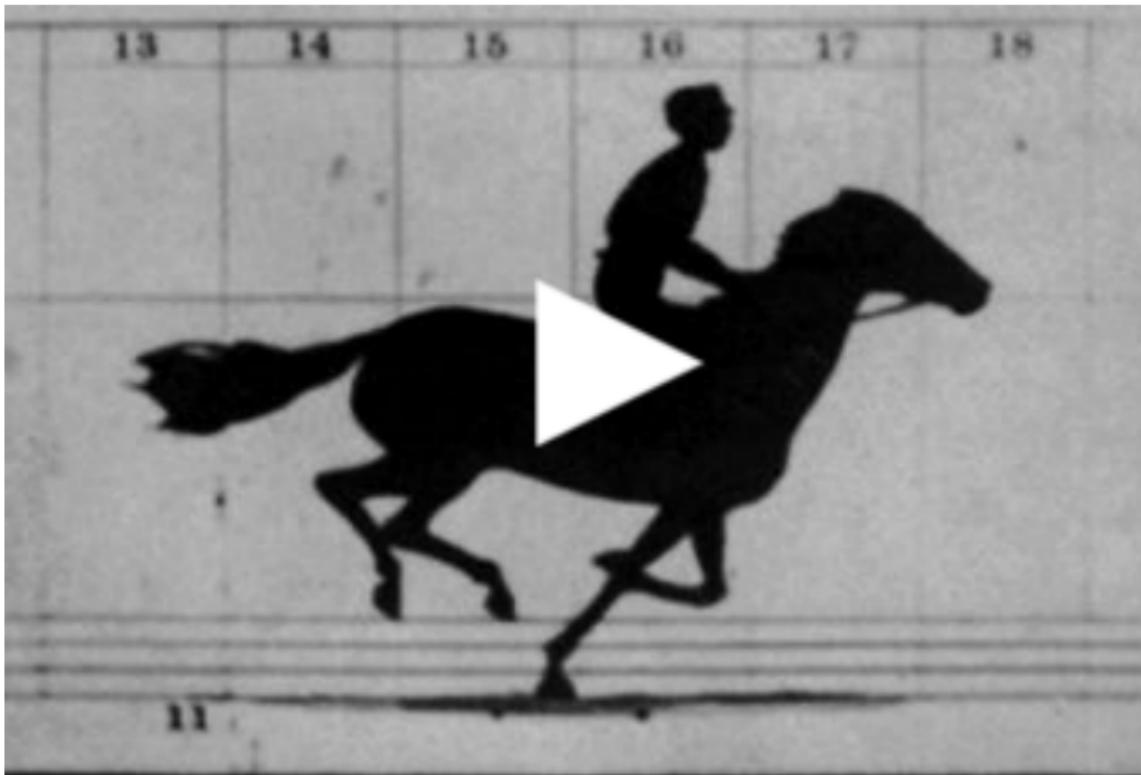


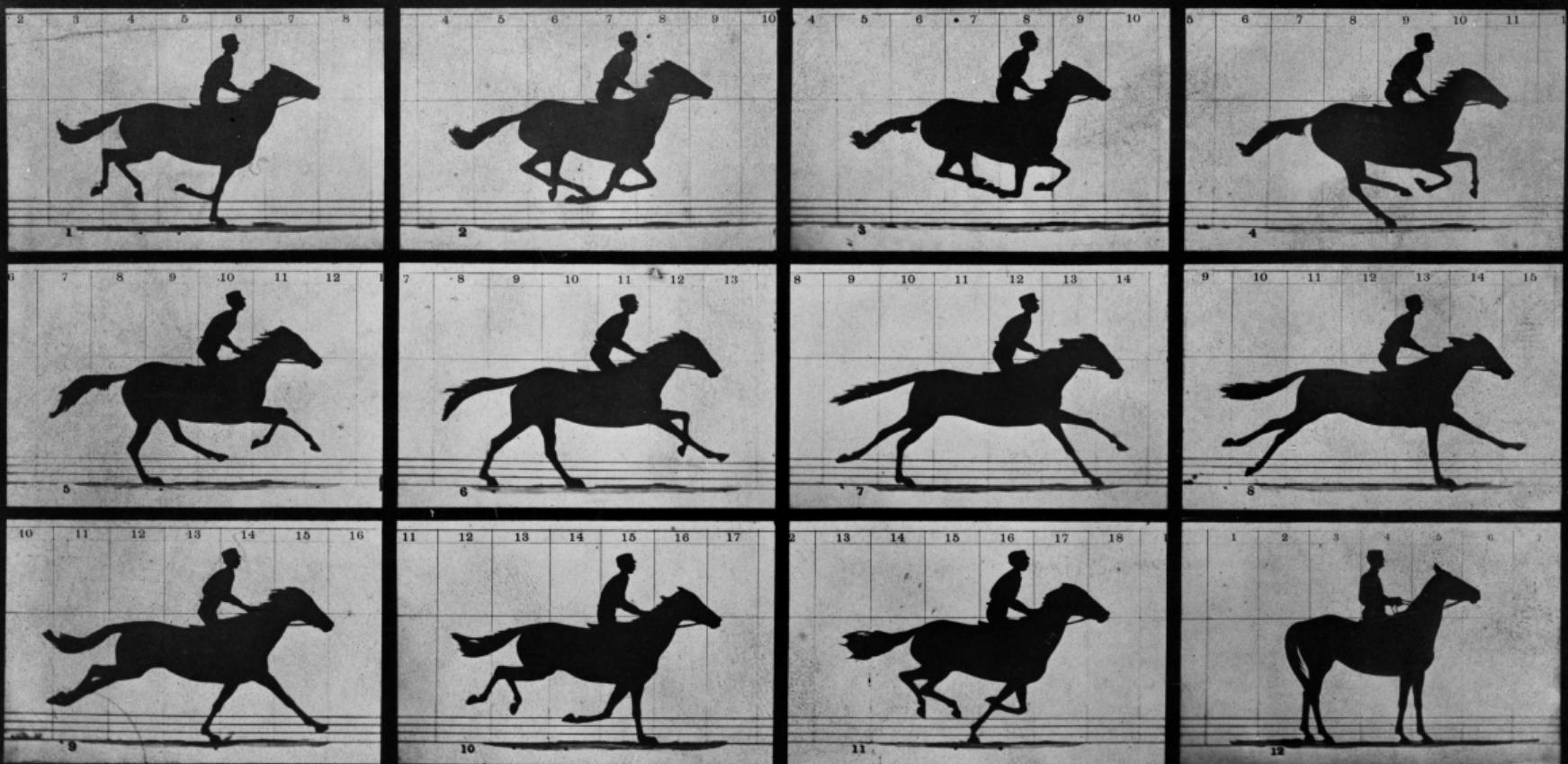
Enclave x-ray: Keymap bit traversal (ground truth)

Sancus IRQ timing attack: Inferring key strokes



Does this also work for Intel SGX enclaves?





Copyright, 1878, by MUYBRIDGE.

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

THE HORSE IN MOTION.

Illustrated by

LEWIS CARROLL

CHARLES L. DODGSON

Building a precise single-stepping primitive



SGX-Step goal: executing enclaves one instruction at a time

Challenge: we need a very precise timer interrupt:

- (:(x86 hardware *debug features* disabled in enclave mode)
- (:) ... but we have *root access!*

Building a precise single-stepping primitive

⚠ SGX-Step goal: executing enclaves one instruction at a time

Challenge: we need a very precise timer interrupt:

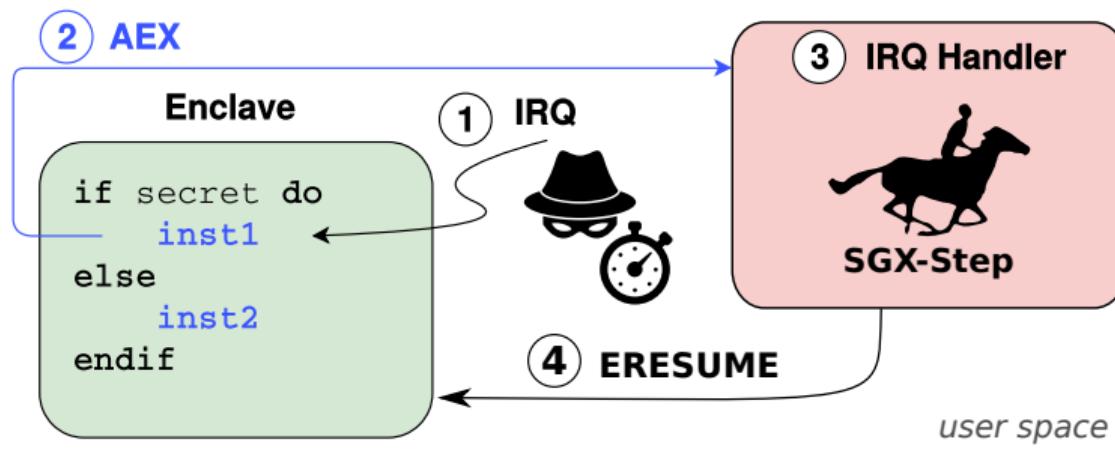
- (:(x86 hardware *debug features* disabled in enclave mode
- (:) ... but we have *root access!*

⇒ Setup user-space virtual **memory mappings** for x86 APIC

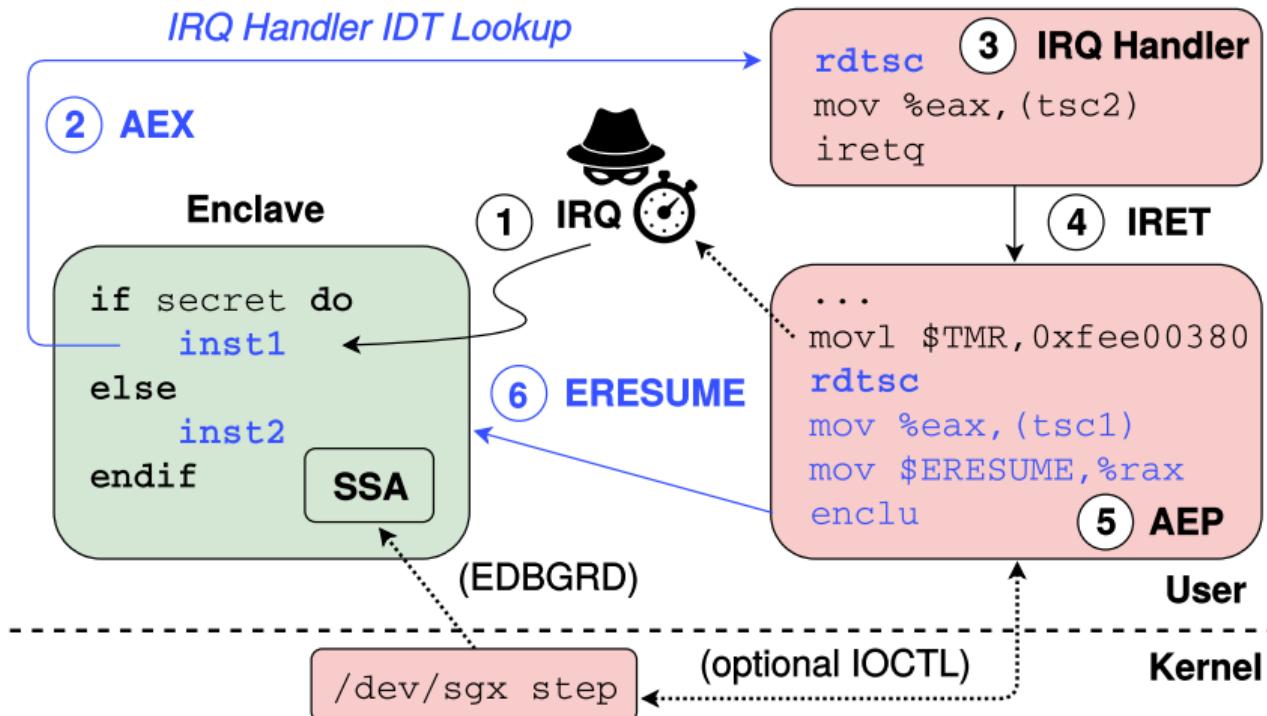
```
jo@sgx-laptop:~$ cat /proc/iomem | grep "Local APIC"
fee00000-fee0ffff : Local APIC
jo@sgx-laptop:~$ sudo devmem2 0xFEE00030 h
/dev/mem opened.
Memory mapped at address 0x7f37dc187000.
Value at address 0xFEE00030 (0x7f37dc187030): 0x15
jo@sgx-laptop:~$ █
```

SGX-Step: Executing enclaves one instruction at a time

User-space attack primitives: APIC timer + interrupt handling 😊

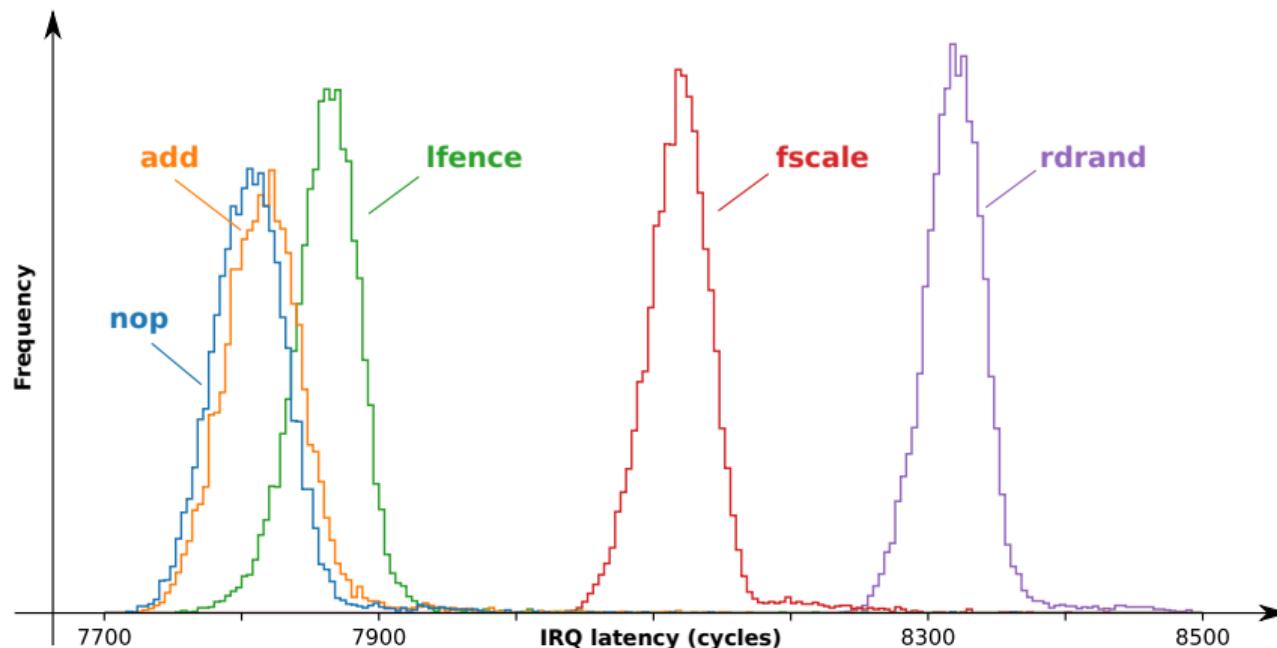


SGX-Step: Executing enclaves one instruction at a time



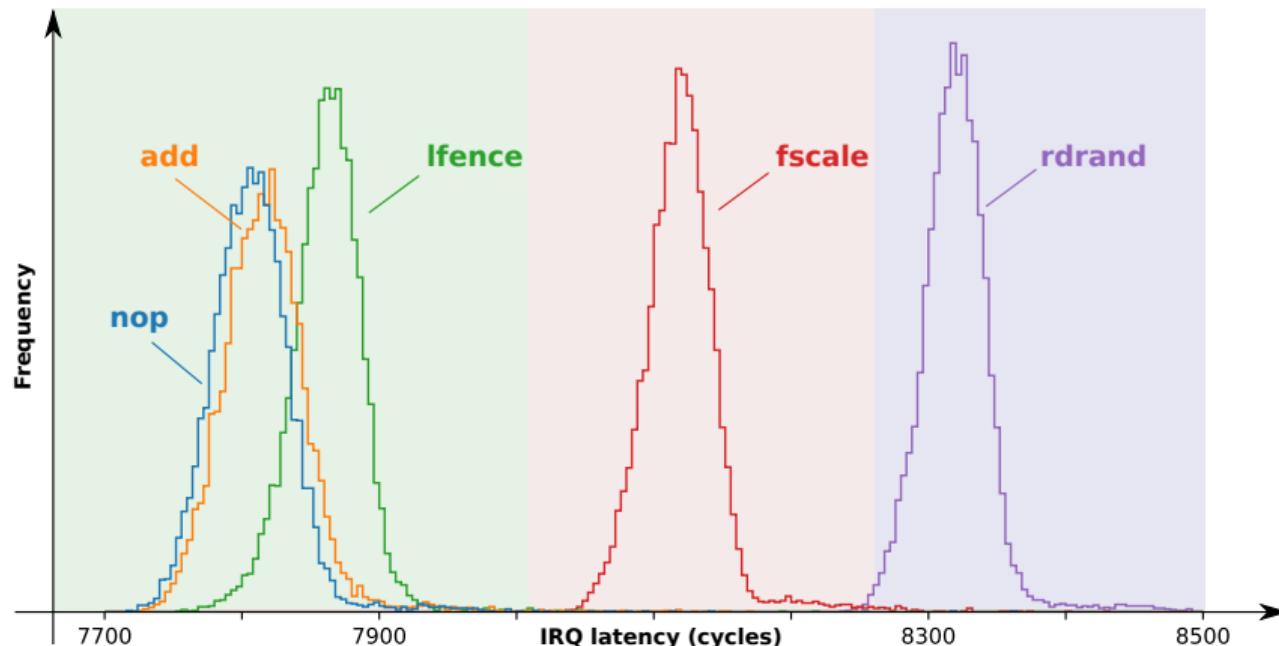
Microbenchmarks: Measuring Intel x86 instruction latencies

Latency distribution: 10,000 samples from benchmark enclave



Microbenchmarks: Measuring Intel x86 instruction latencies

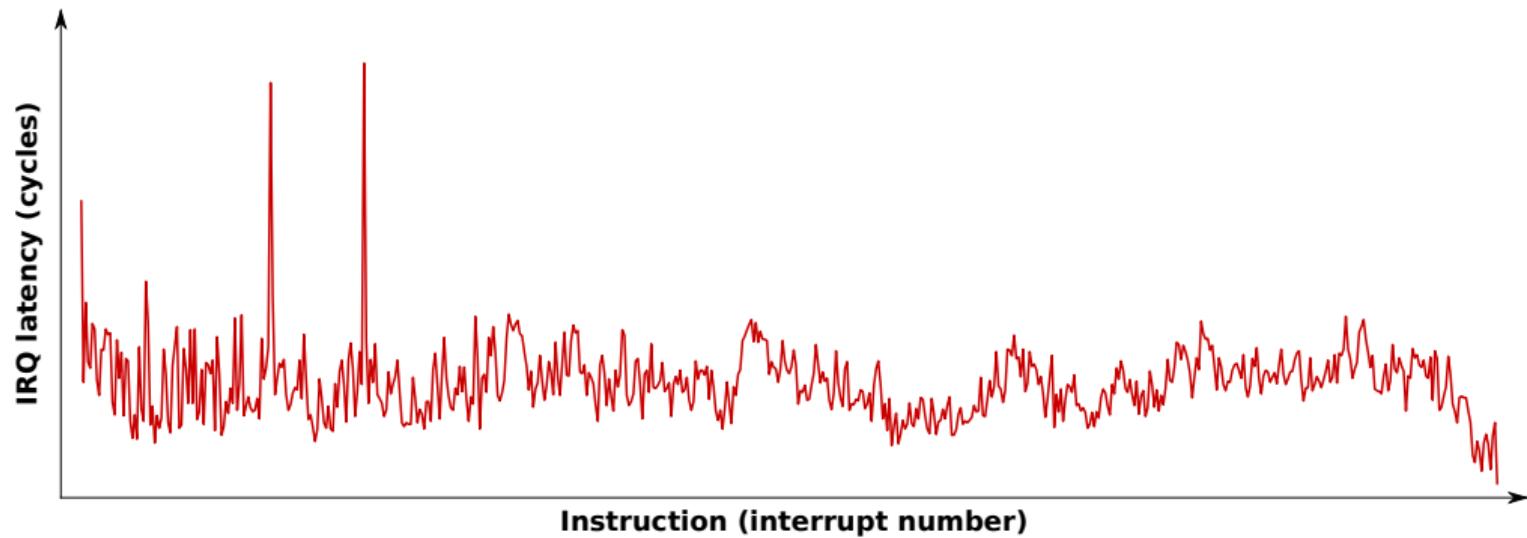
Timing leak: reconstruct *instruction latency class*



Single-stepping Intel SGX enclaves in practice



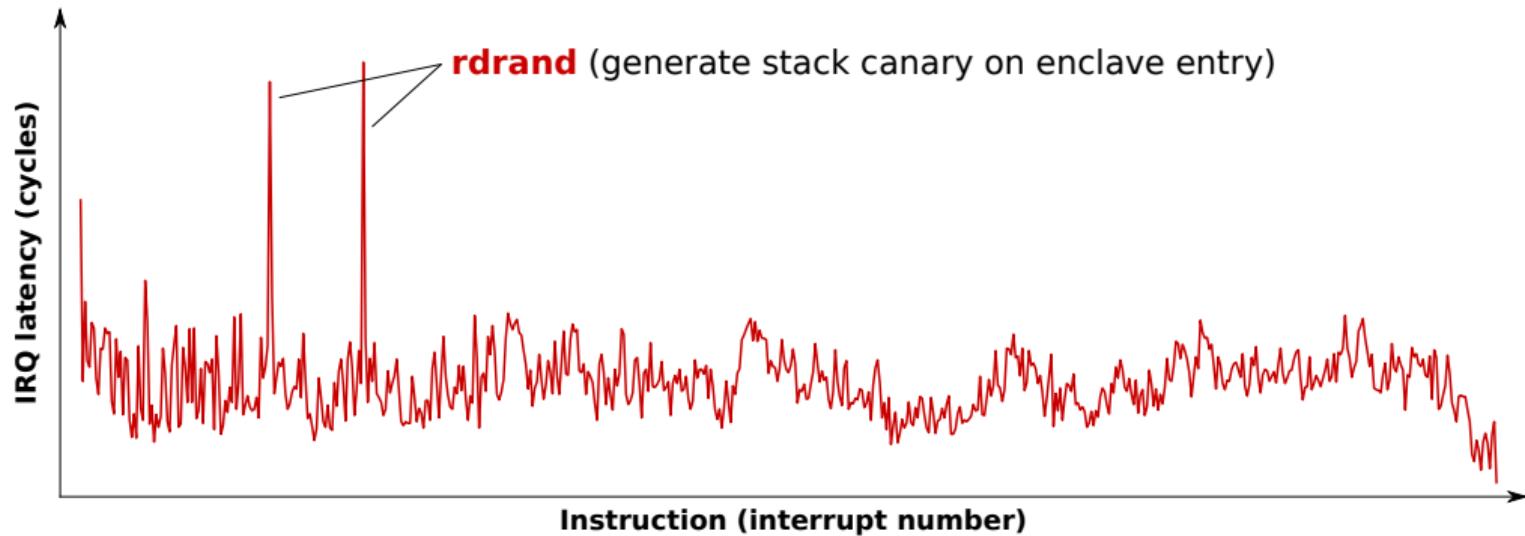
Enclave x-ray: Start-to-end trace enclaved execution



Single-stepping Intel SGX enclaves in practice



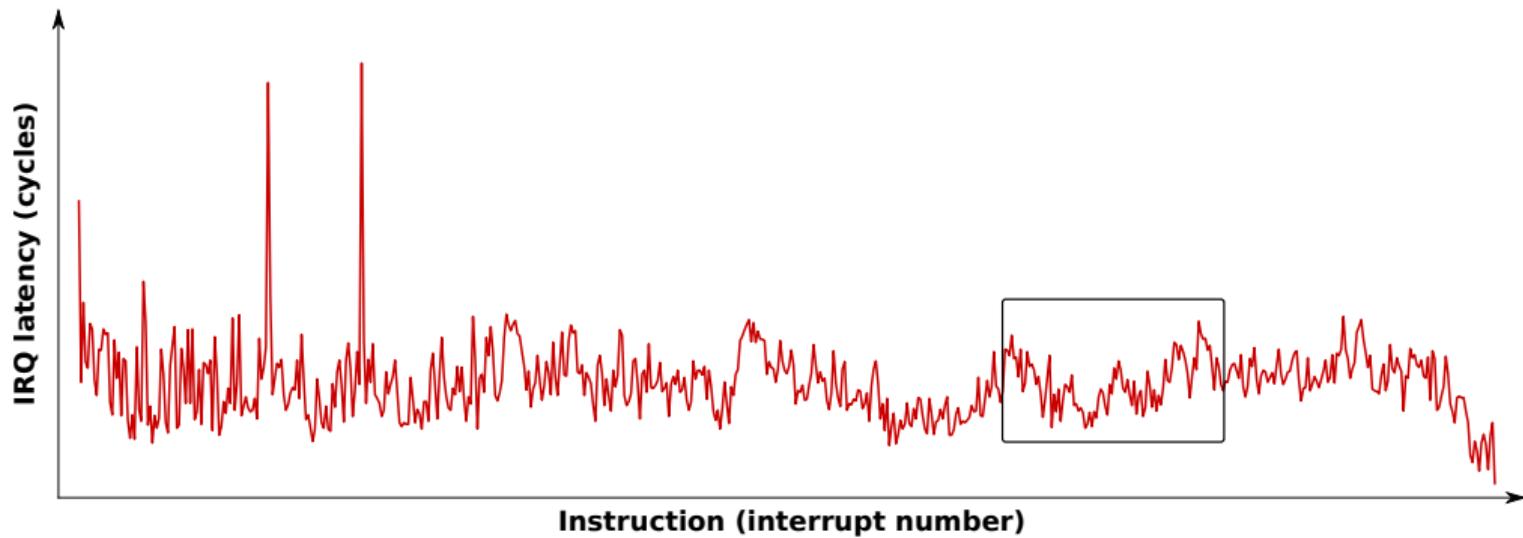
Enclave x-ray: Spotting high-latency instructions



Single-stepping Intel SGX enclaves in practice

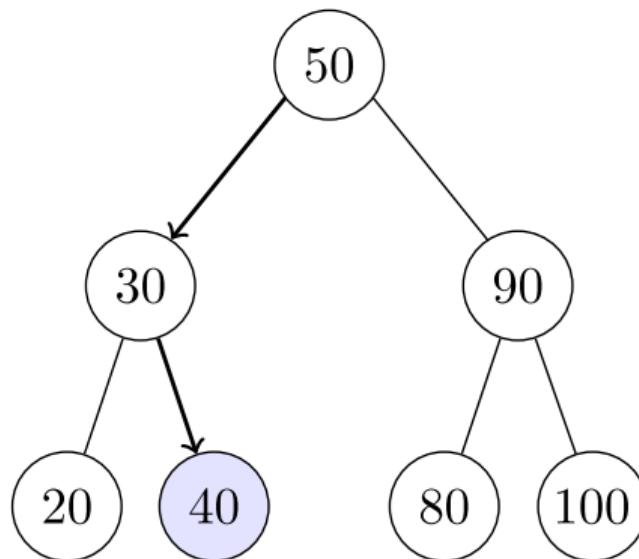


Enclave x-ray: Zooming in on bsearch function



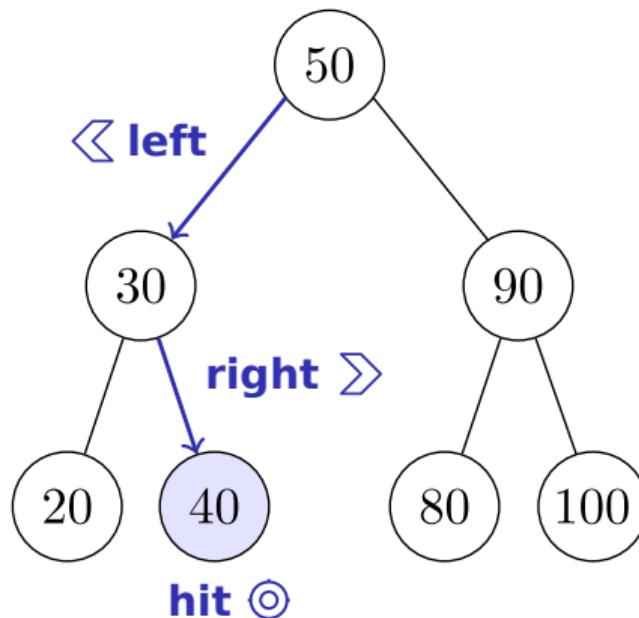
De-anonymizing enclave lookups with interrupt latency

Binary search: Find 40 in {20, 30, 40, 50, 80, 90, 100}



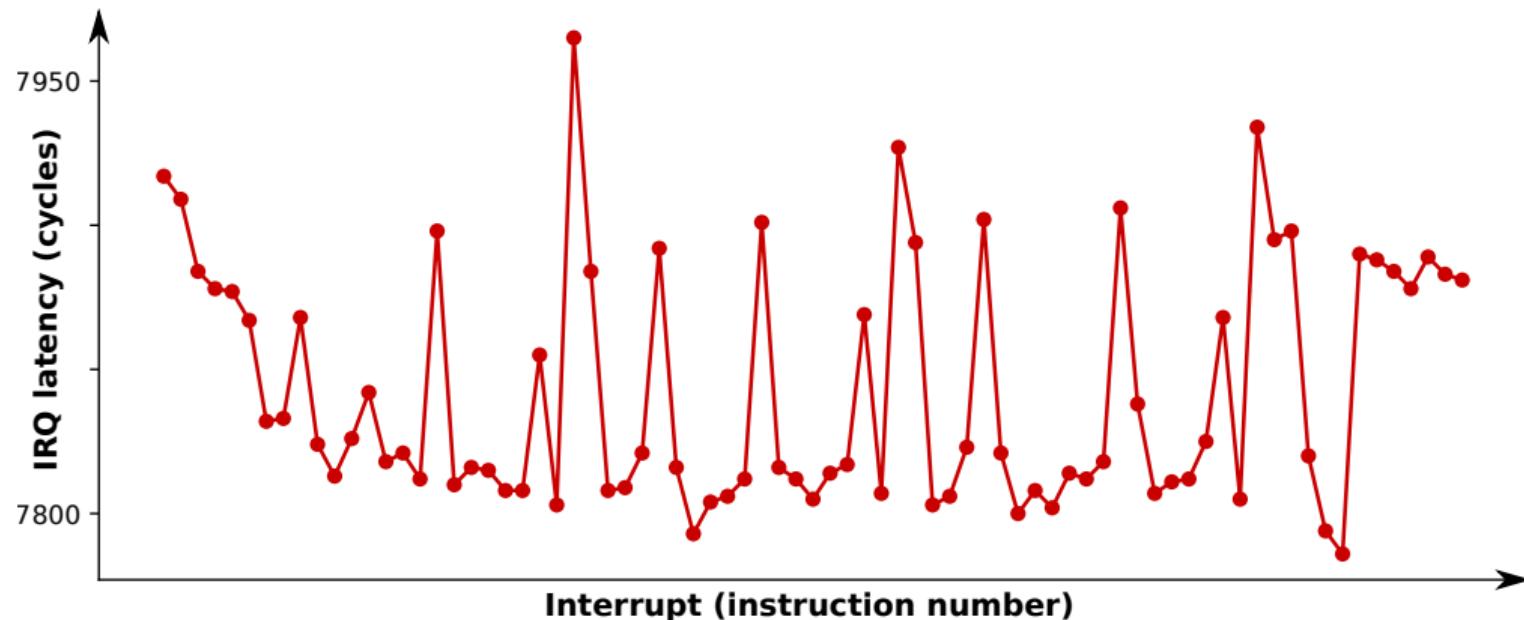
De-anonymizing enclave lookups with interrupt latency

Adversary: Infer secret lookup in known array



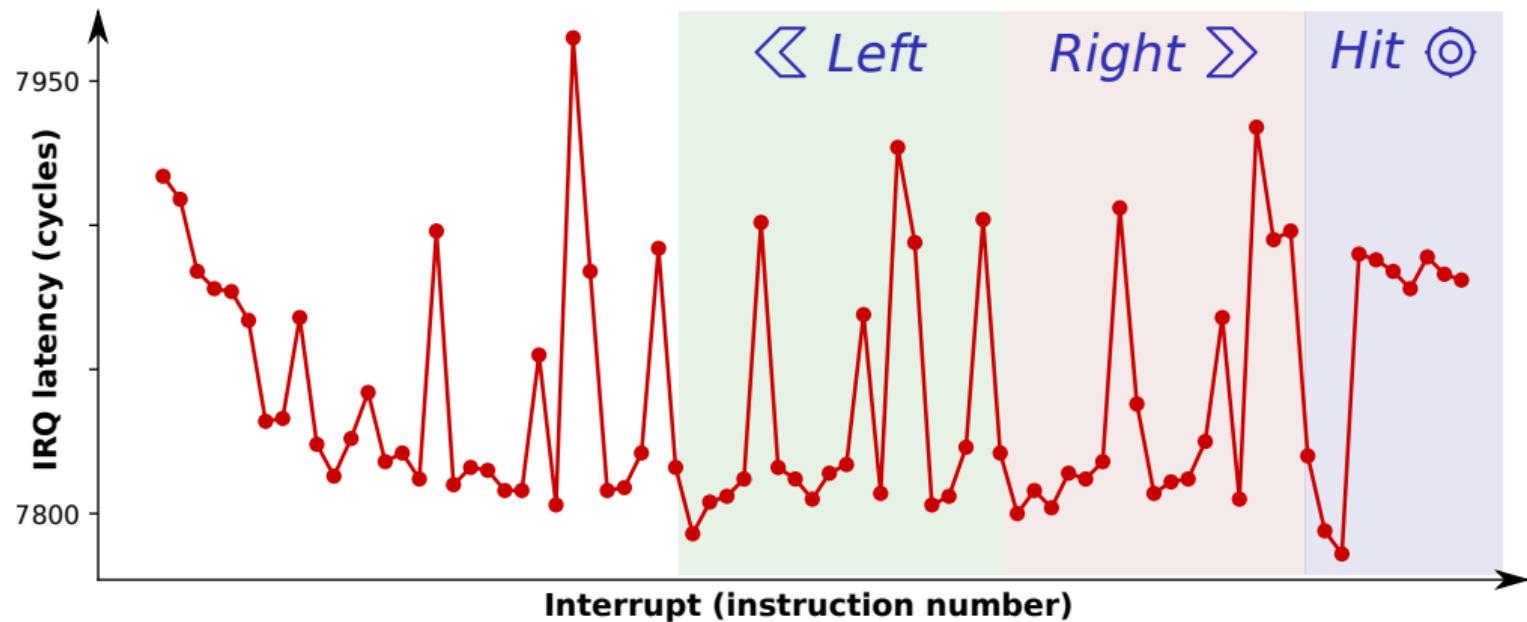
De-anonymizing enclave lookups with interrupt latency

Goal: Infer lookup → reconstruct bsearch control flow



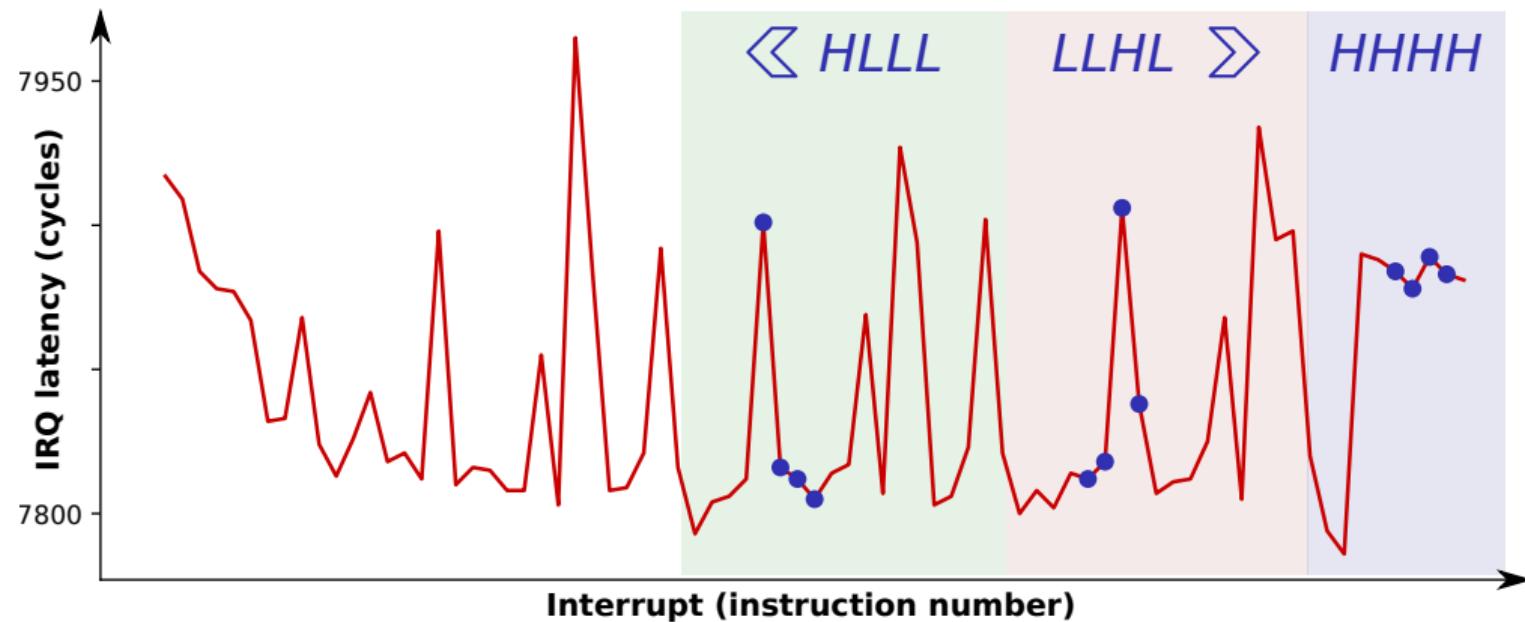
De-anonymizing enclave lookups with interrupt latency

Goal: Infer lookup → reconstruct bsearch control flow



De-anonymizing enclave lookups with interrupt latency

⇒ Sample **instruction latencies** in secret-dependent path

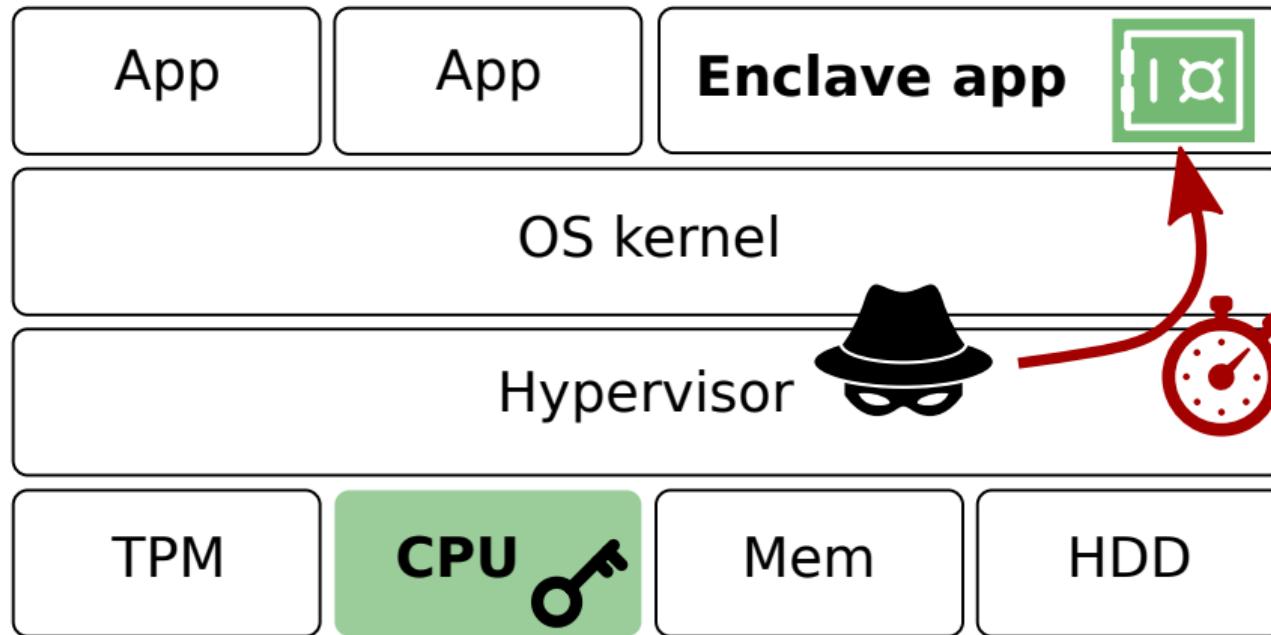


Privileged adversary idea #3



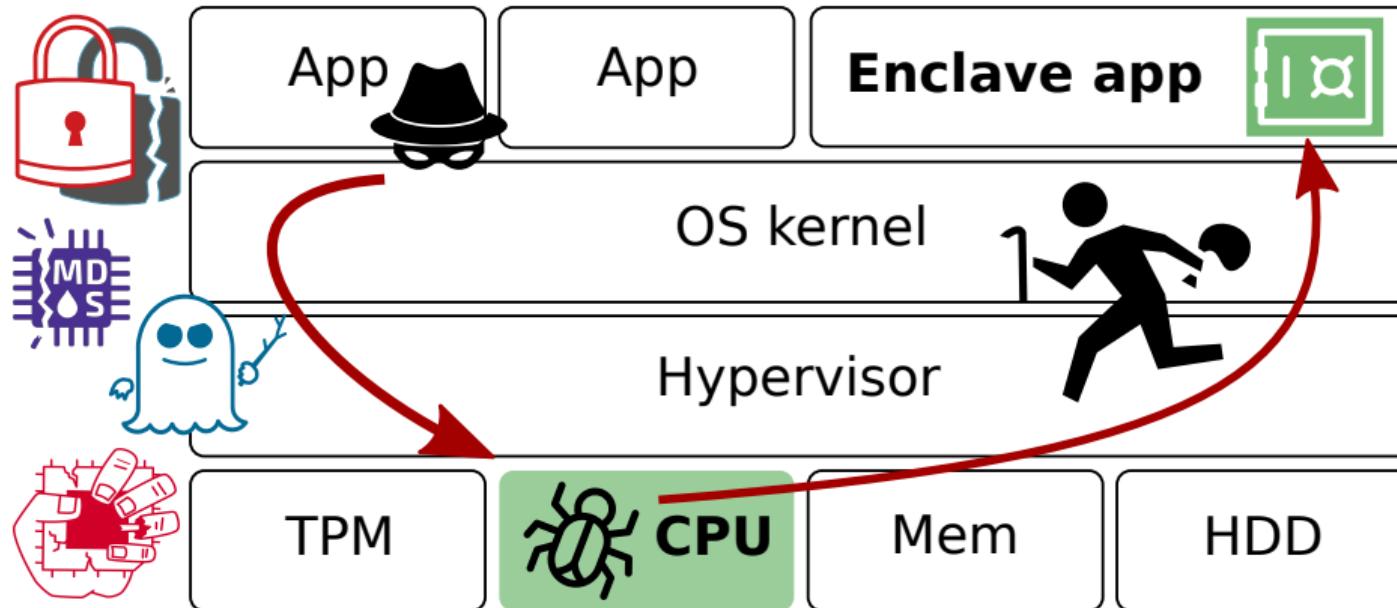
Page tables revisited: transient execution?

Enclaved execution: Side-channel attacks



Untrusted OS → new class of powerful **side channels**

Enclaved execution: Transient-execution attacks



Trusted CPU → exploit **microarchitectural bugs/design flaws**

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.

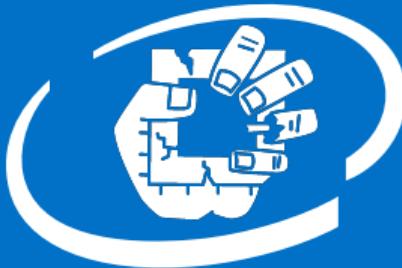




inside™



inside™

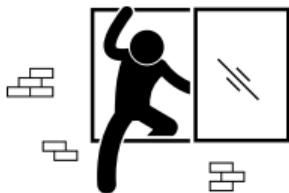


inside™



inside™

Meltdown: Transiently encoding unauthorized memory



Unauthorized access

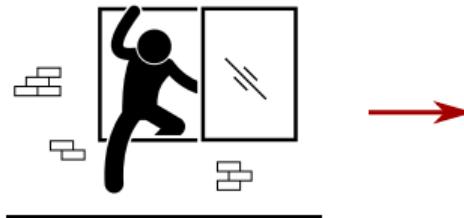
Listing 1: x86 assembly

```
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

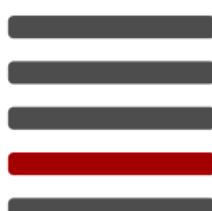
Listing 1: x86 assembly.

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

Listing 2: C code.

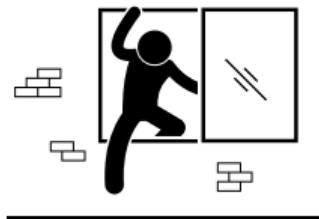
```
1 void meltdown(  
2     uint8_t *oracle,  
3     uint8_t *secret_ptr)  
4 {  
5     uint8_t v = *secret_ptr;  
6     v = v * 0x1000;  
7     uint64_t o = oracle[v];  
8 }
```

oracle array



secret idx

Meltdown: Transiently encoding unauthorized memory



Unauthorized access



Transient out-of-order window



Exception

(discard architectural state)

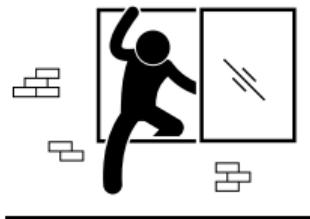
Listing 1: x86 assembly.

```
1 meltdown:  
2 // %rdi: oracle  
3 // %rsi: secret_ptr  
4  
5 movb (%rsi), %al  
6 shr $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

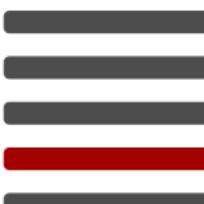
Listing 1: x86 assembly.

```
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



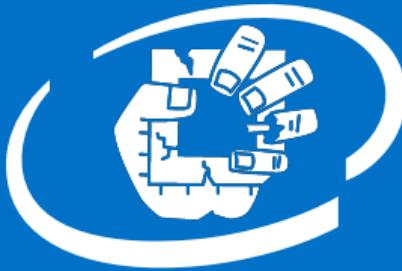
cache hit



inside™



inside™



inside™



inside™

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

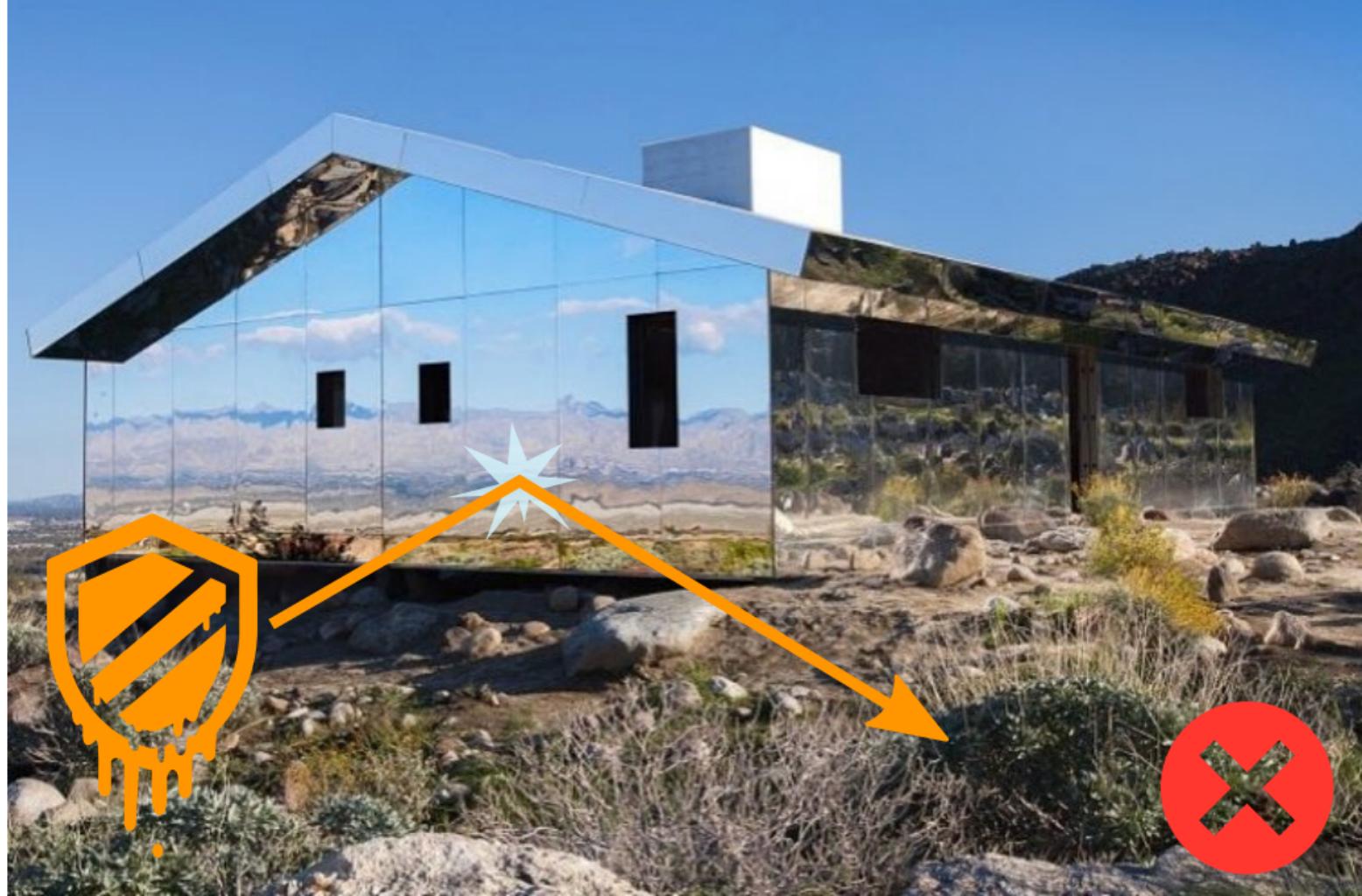
I'M SURE THIS WON'T BE THE LAST SUCH PROBLEM —

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.

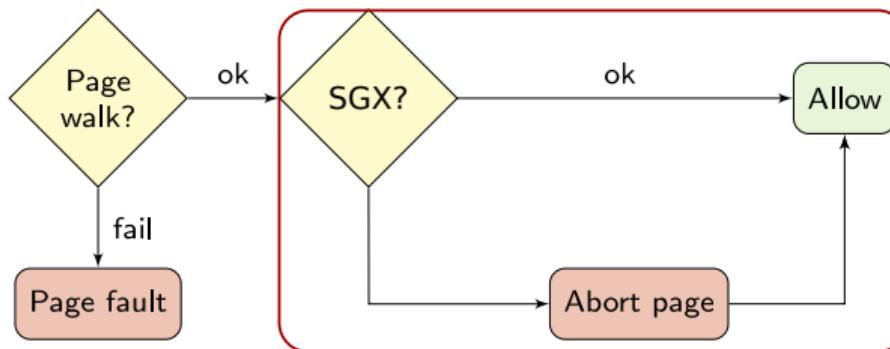
<https://wired.com> and <https://arstechnica.com>





Building Foreshadow: Evade SGX abort page semantics

Note: SGX MMU sanitizes *untrusted* address translation

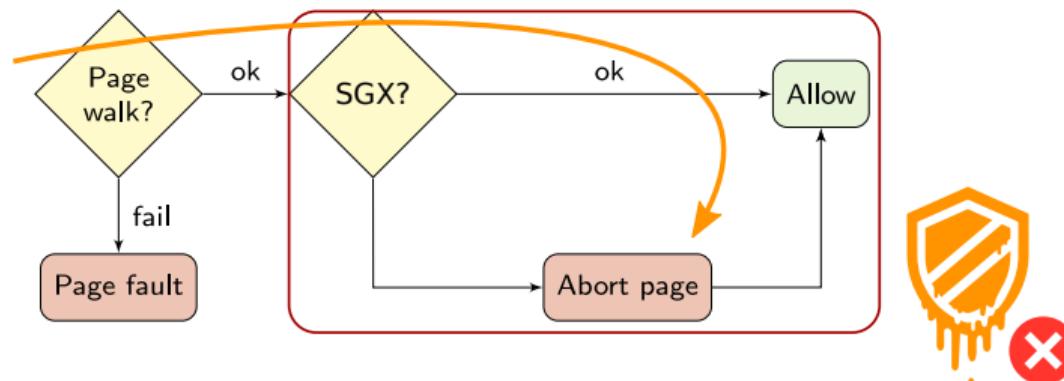


Abort page semantics:

An attempt to read from a non-existent or disallowed resource returns all ones for data (abort page). An attempt to write to a non-existent or disallowed physical resource is dropped. This behavior is unrelated to exception type abort (the others being Fault and Trap).

Building Foreshadow: Evade SGX abort page semantics

Straw man: (Transient) accesses in non-enclave mode are dropped



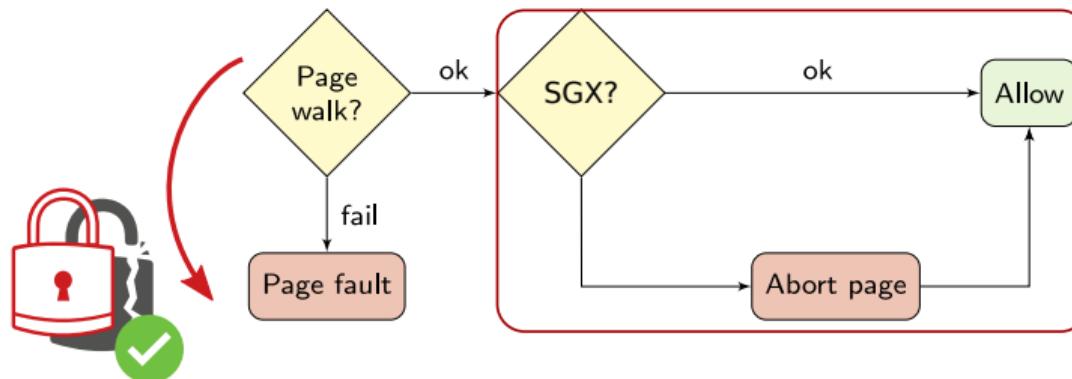
Abort page semantics:

An attempt to read from a non-existent or disallowed resource returns all ones for data (abort page). An attempt to write to a non-existent or disallowed physical resource is dropped. This behavior is unrelated to exception type abort (the others being Fault and Trap).

<https://software.intel.com/en-us/sgx-sdk-dev-reference-enclave-development-basics>

Building Foreshadow: Evade SGX abort page semantics

Stone man: Bypass abort page via *untrusted* page table

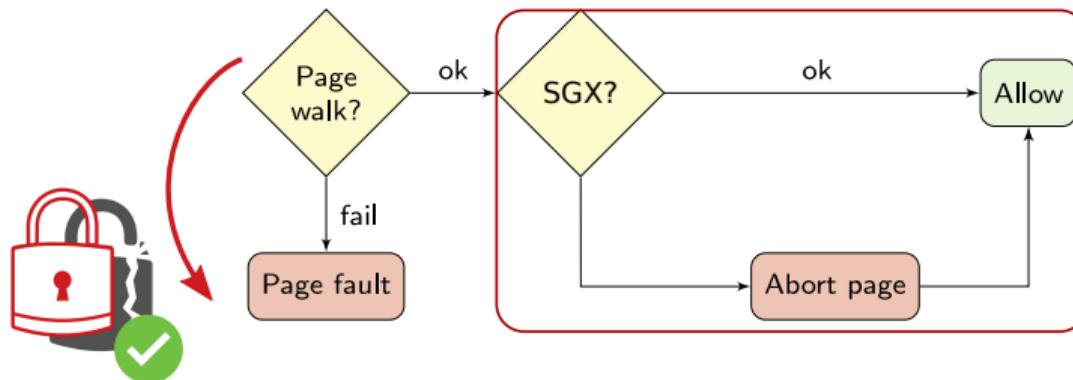


Xu et al. "Controlled-channel attacks: Deterministic side channels for untrusted operating systems", IEEE S&P 2015

Van Bulck et al. "Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution", USENIX 2017

Building Foreshadow: Evade SGX abort page semantics

Stone man: Bypass abort page via *untrusted* page table

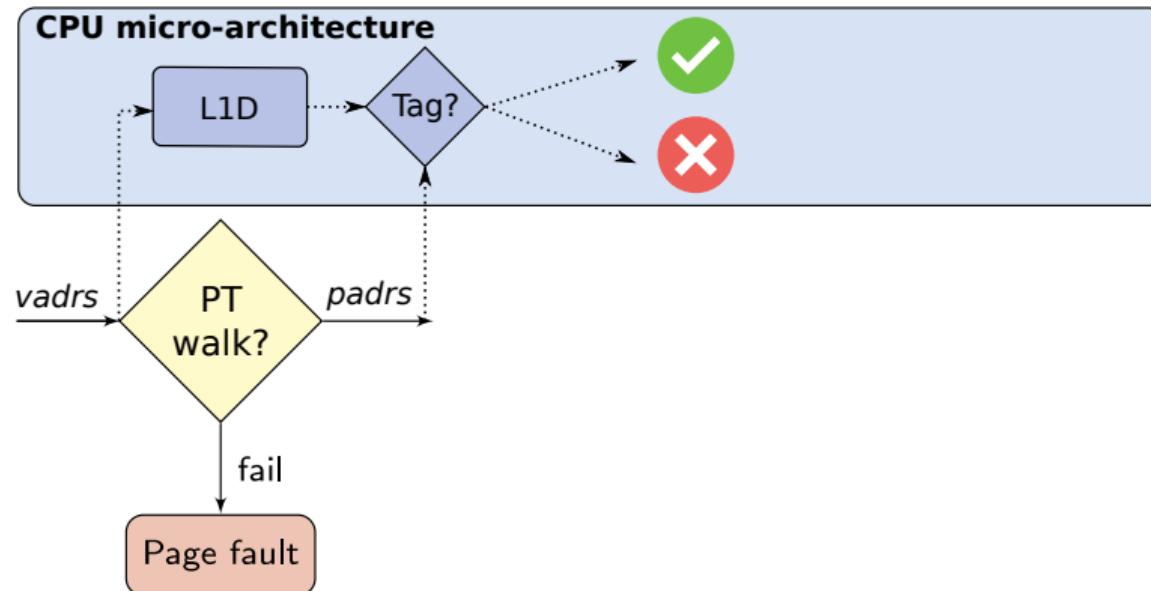


Unprivileged system call

```
mprotect( secret_ptr & 0xFFF, 0x1000, PROT_NONE );
```

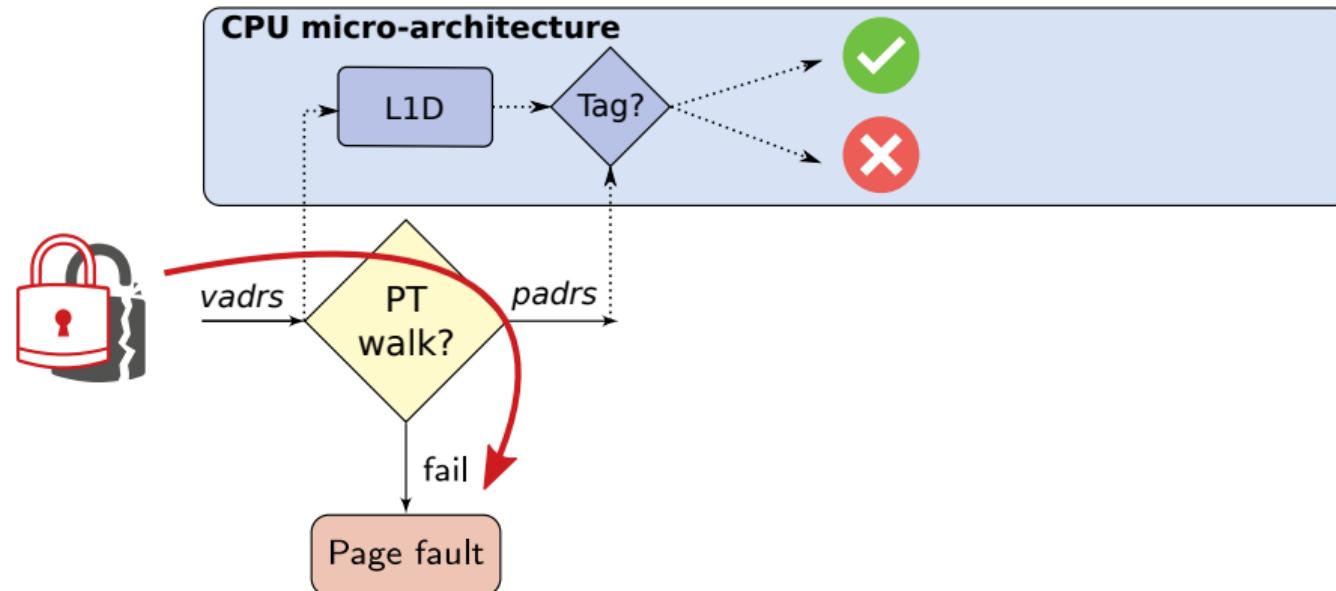


Foreshadow-NG: Breaking the virtual memory abstraction



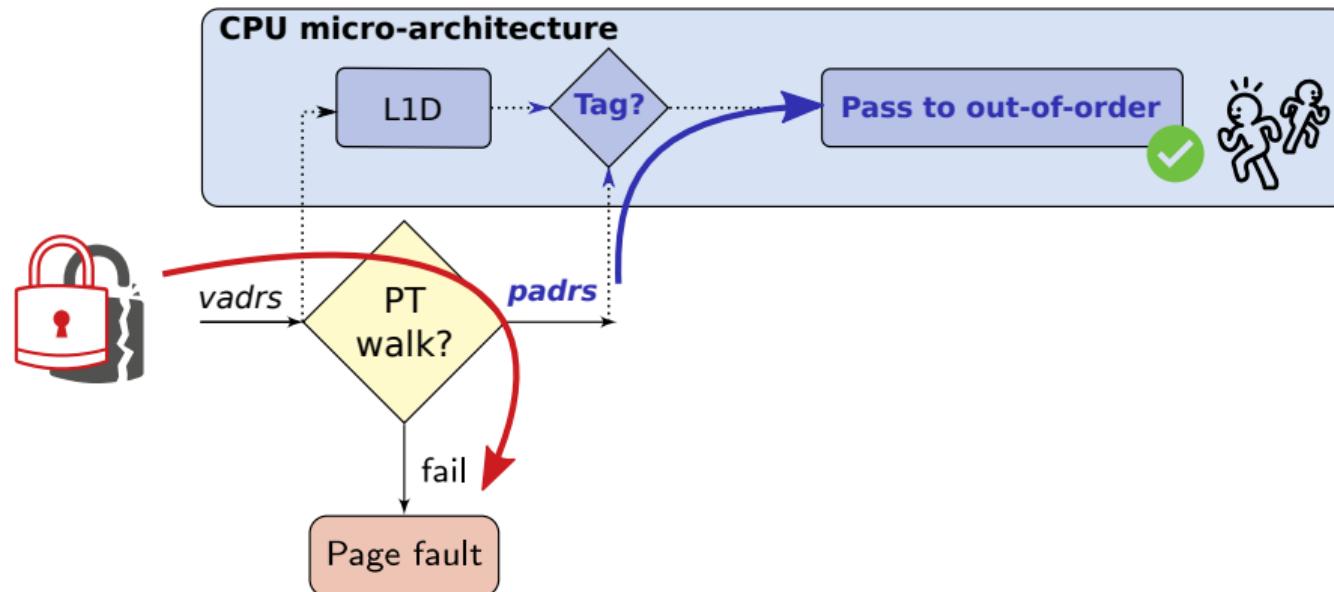
L1 cache design: Virtually-indexed, physically-tagged

Foreshadow-NG: Breaking the virtual memory abstraction



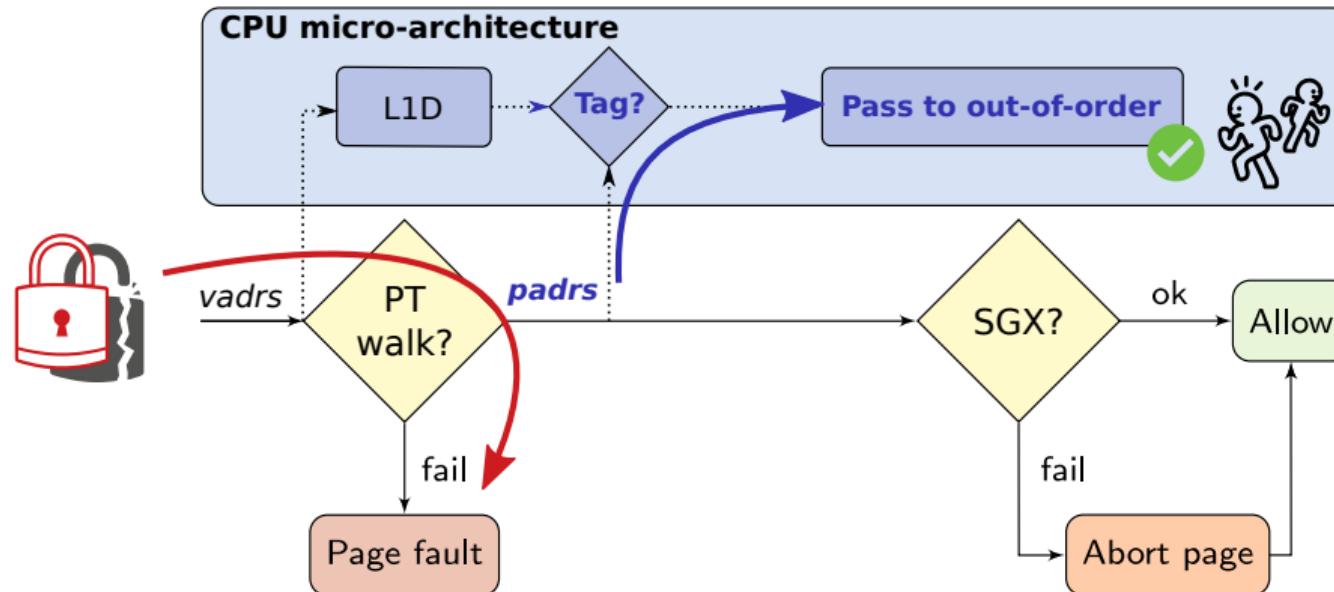
Page fault: Early-out address translation

Foresight-NG: Breaking the virtual memory abstraction



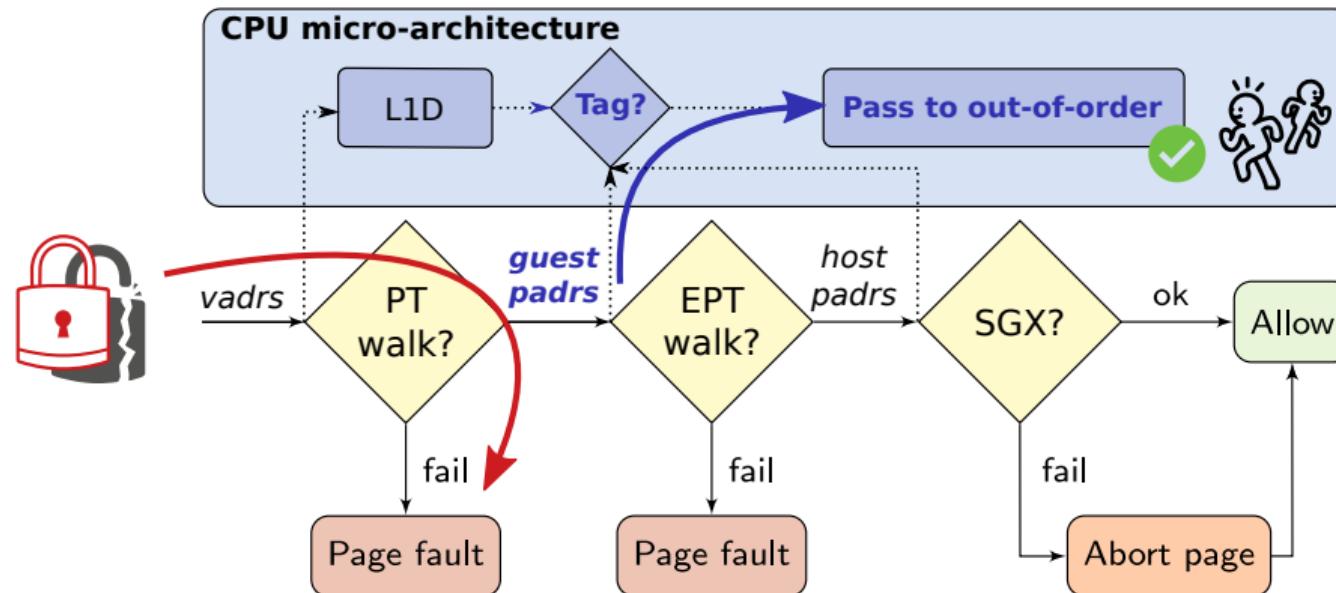
L1-Terminal Fault: match *unmapped physical address* (!)

Foresight-NG: Breaking the virtual memory abstraction

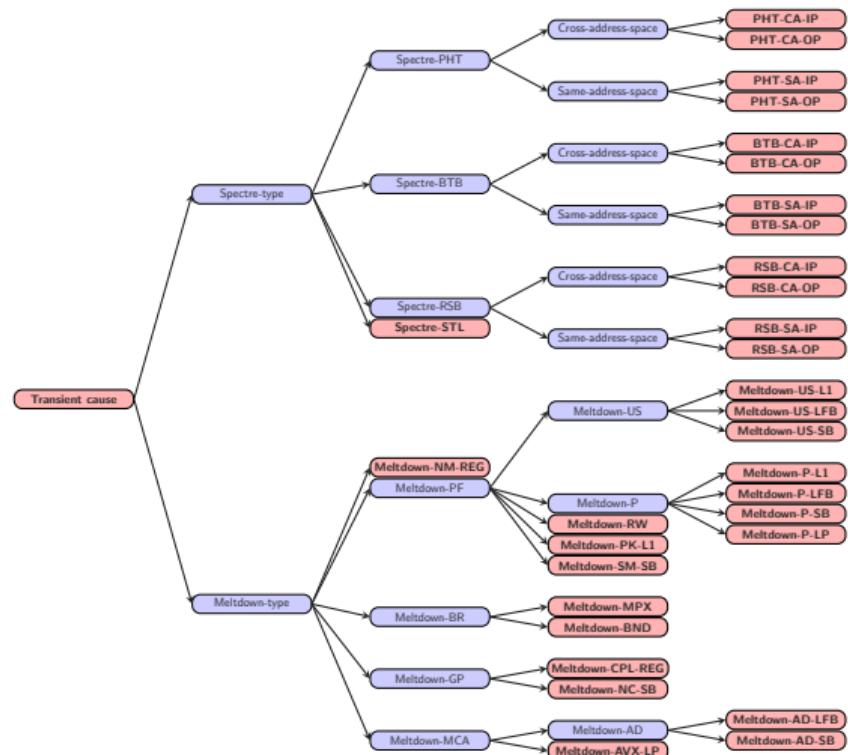


Foresight-NG: bypass enclave isolation

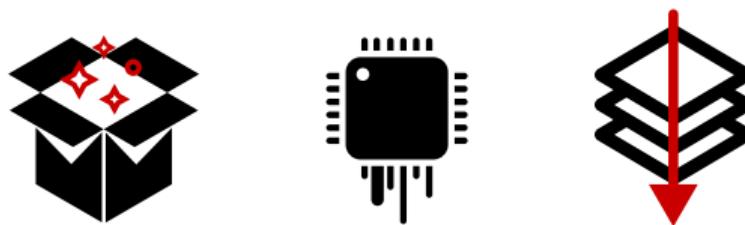
Foreshadow-NG: Breaking the virtual memory abstraction



Foreshadow-VMM: bypass virtual machine isolation



- ⇒ Trusted execution environments are not perfect(!)
- ⇒ New emerging and powerful class of transient-execution attacks
- ⇒ Importance of fundamental side-channel research; no silver-bullet defenses



Appendix

References i

-  C. Canella, J. Van Bulck, M. Schwarz, M. Lipp, B. von Berg, P. Ortner, F. Piessens, D. Evtyushkin, and D. Gruss.
A systematic evaluation of transient execution attacks and defenses.
arXiv preprint arXiv:1811.05441, 2018.
-  J. Noorman, J. Van Bulck, J. T. Mühlberg, F. Piessens, P. Maene, B. Preneel, I. Verbauwhede, J. Götzfried, T. Müller, and F. Freiling.
Sancus 2.0: A low-cost security architecture for IoT devices.
ACM Transactions on Privacy and Security (TOPS), 20(3):7:1–7:33, 2017.

References ii

-  M. Schwarz, M. Lipp, D. Moghimi, J. Van Bulck, J. Stecklina, T. Prescher, and D. Gruss.
ZombieLoad: Cross-privilege-boundary data sampling.
In *CCS*, 2019.
-  J. Van Bulck, J. T. Mühlberg, and F. Piessens.
VulCAN: Efficient component authentication and software isolation for automotive control networks.
In *Annual Computer Security Applications Conference (ACSAC)*, 2017.

References iii

-  J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx.

Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution.

In *Proceedings of the 27th USENIX Security Symposium*, 2018.

-  J. Van Bulck, F. Piessens, and R. Strackx.

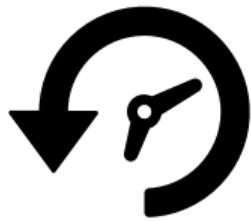
SGX-Step: A practical attack framework for precise enclave execution control.

In *SysTEX*, pp. 4:1–4:6, 2017.

References iv

-  J. Van Bulck, F. Piessens, and R. Strackx.
Nemesis: Studying microarchitectural timing leaks in rudimentary cpu interrupt logic.
In *ACM CCS 2018*, 2018.
-  J. Van Bulck, N. Weichbrodt, R. Kapitza, F. Piessens, and R. Strackx.
Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution.
In *Proceedings of the 26th USENIX Security Symposium*, pp. 1041–1056, 2017.

Mitigating Foreshadow



1. Cache secrets in L1

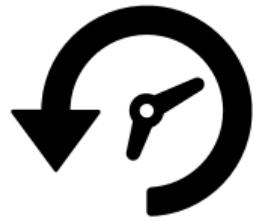


2. Unmap [page table](#) entry



3. Execute [Meltdown](#)

Mitigating Foreshadow



1. Cache secrets in L1



2. Unmap page table entry



3. Execute Meltdown

Future CPUs
(silicon-based changes)

Mitigating Foreshadow



1. Cache secrets in L1



2. Unmap page table entry



3. Execute Meltdown

OS kernel updates
(sanitize page frame bits)

Mitigating Foreshadow



1. Cache secrets in L1



2. Unmap page table entry

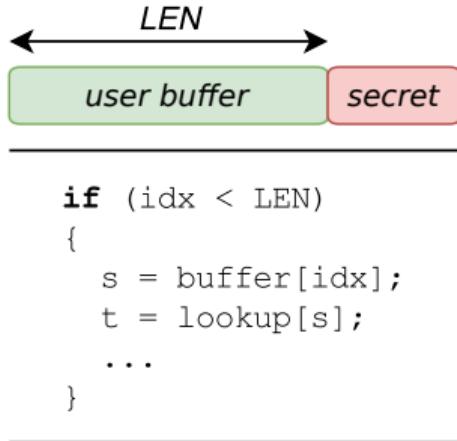


3. Execute Meltdown

Intel microcode updates

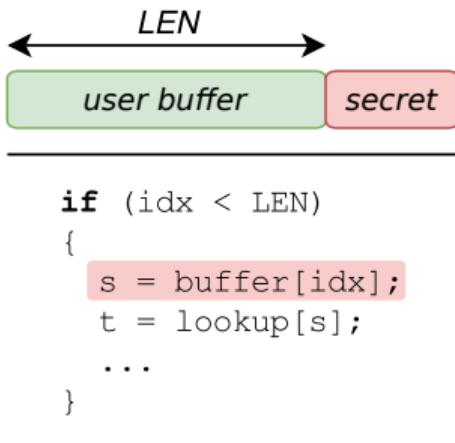
⇒ **Flush L1** cache on enclave/VMM exit + **disable HyperThreading**

Spectre v1: Speculative buffer over-read



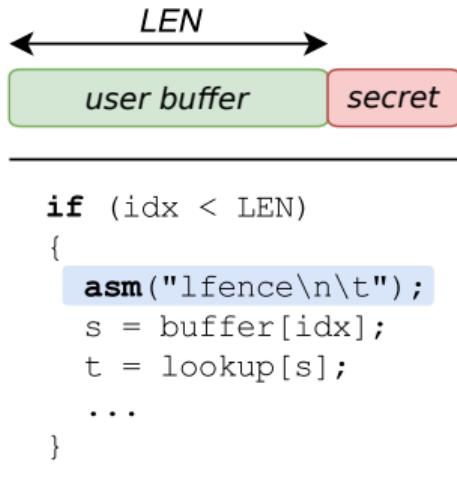
- Programmer *intention*: never access out-of-bounds memory

Spectre v1: Speculative buffer over-read



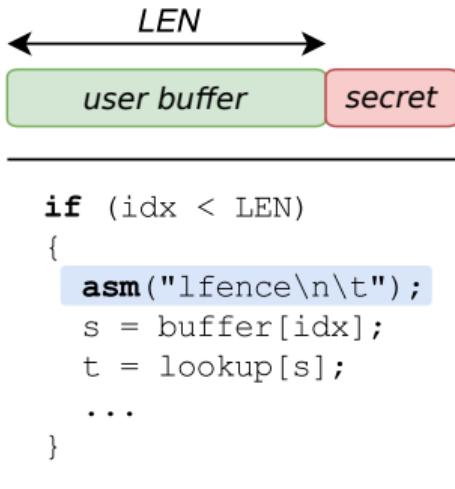
- Programmer *intention*: never access out-of-bounds memory
- Branch can be mistrained to **speculatively** (i.e., ahead of time) execute with $idx \geq LEN$ in the **transient world**

Spectre v1: Speculative buffer over-read



- Programmer *intention*: never access out-of-bounds memory
- Branch can be mistrained to **speculatively** (i.e., ahead of time) execute with $idx \geq LEN$ in the **transient world**
- Insert explicit **speculation barriers** to tell the CPU to halt the transient world...

Spectre v1: Speculative buffer over-read



- Programmer *intention*: never access out-of-bounds memory
- Branch can be mistrained to **speculatively** (i.e., ahead of time) execute with $idx \geq LEN$ in the **transient world**
- Insert explicit **speculation barriers** to tell the CPU to halt the transient world...
- Huge manual, error-prone effort...