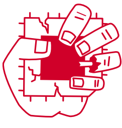
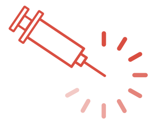
Microarchitectural Side-Channel Attacks for Untrusted Operating Systems

Jo Van Bulck

LSDS seminar, Imperial College London (online), October 29, 2020

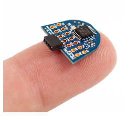
imec-DistriNet, KU Leuven jo.vanbulck@cs.kuleuven.be jovanbulck

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

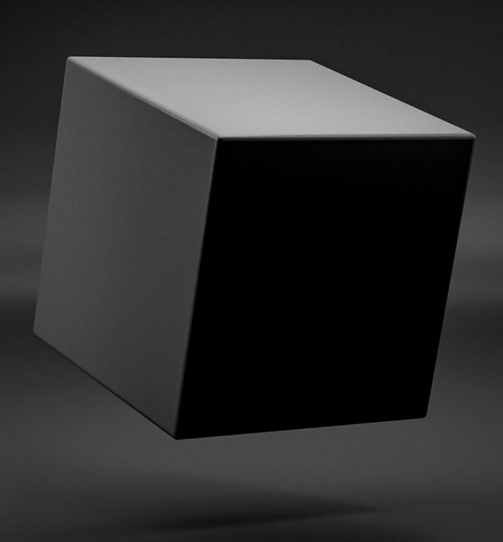
Transient-execution attacks

Side-channel attacks



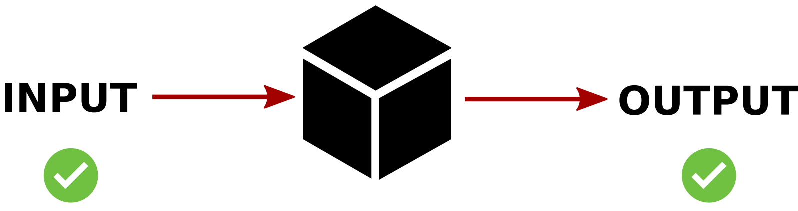
Sancus TEE processor

~40 years of computer security research in one picture

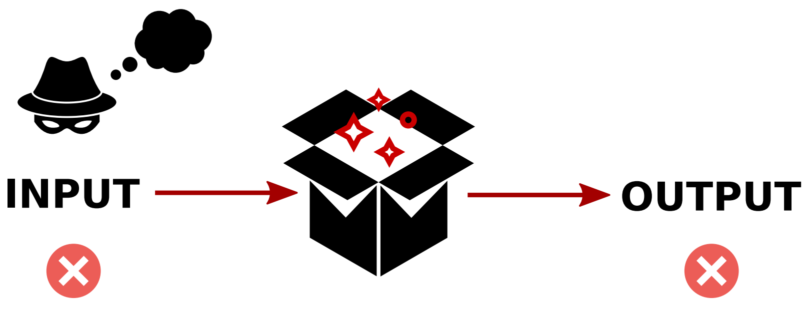


A primer on software security

Secure program: convert all input to *expected output*

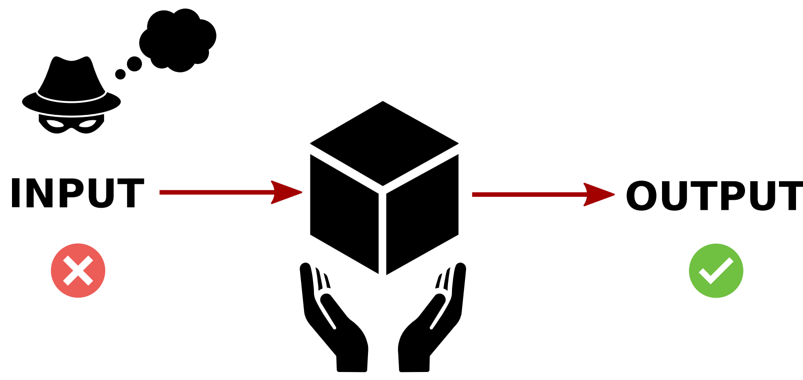


Buffer overflow vulnerabilities: trigger *unexpected behavior*

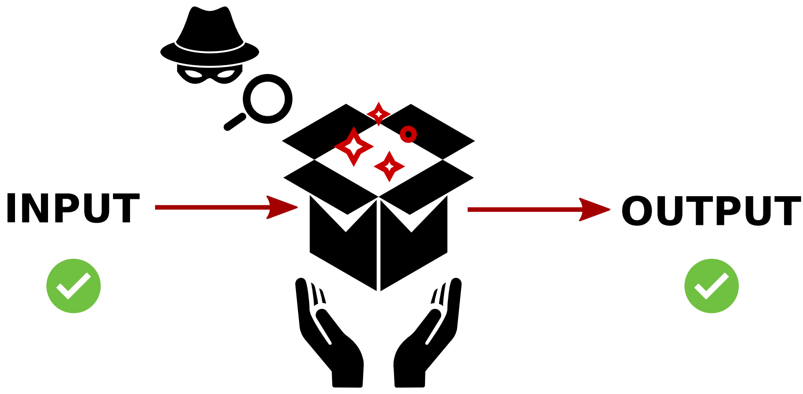


A primer on software security

Safe languages & formal verification: preserve *expected behavior*

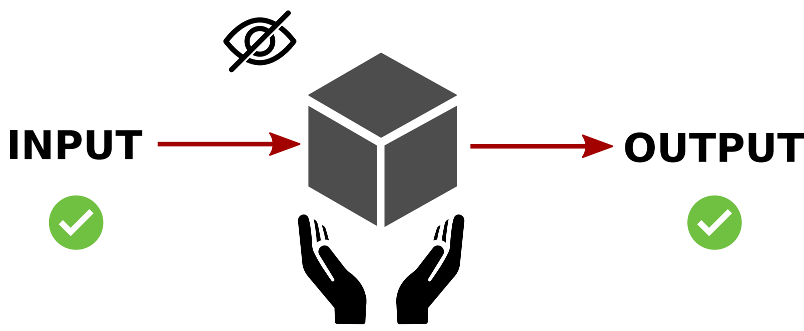


Side-channels: observe *side-effects* of the computation



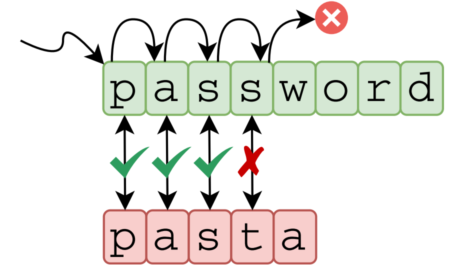
A primer on software security

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 }
```

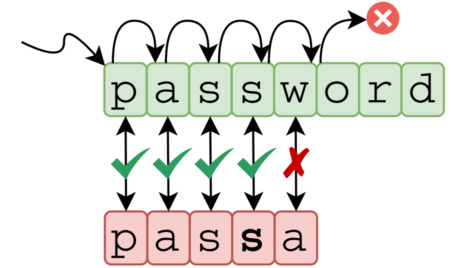


Overall execution time reveals correctness of individual password bytes!

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

A vulnerable example program and its constant-time equivalent

```c
void check_pwd(char *input)
{
    for (int i=0; i < PWD_LEN; i++)
        if (input[i] != pwd[i])
            return 0;

    return 1;
}
```



Overall execution time reveals correctness of individual password bytes!

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

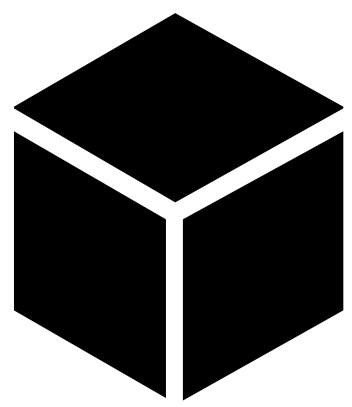
A vulnerable example program and its constant-time equivalent

```c
void check_pwd(char *input)
{
    for (int i=0; i < PWD_LEN; i++)
      if (input[i] != pwd[i])
        return 0;

    return 1;
}
```

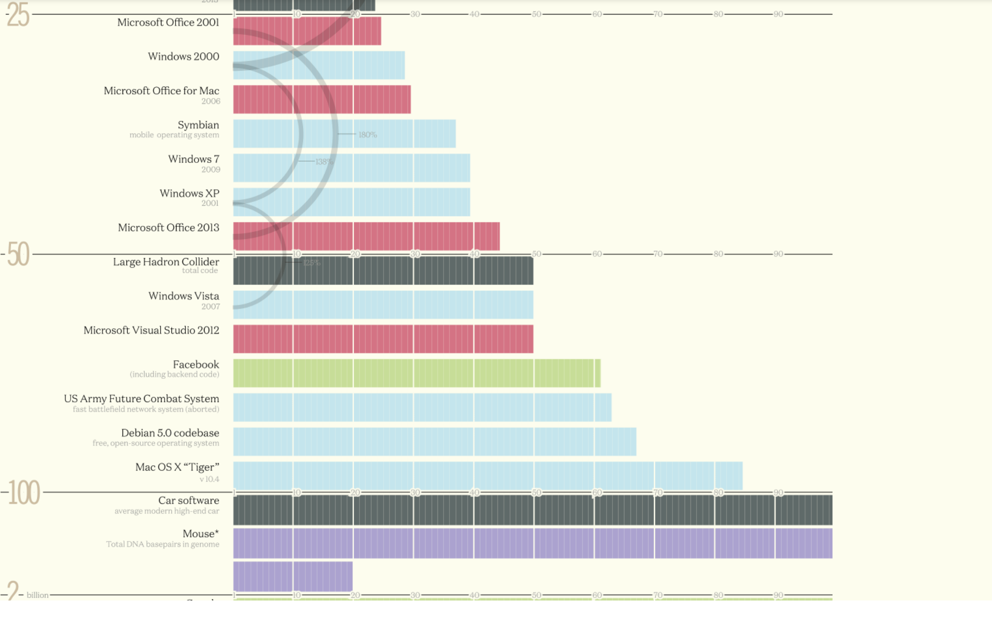
```c
void check_pwd(char *input)
{
    int rv = 0x0;
    for (int i=0; i < PWD_LEN; i++)
        rv |= input[i] ^ pwd[i];

    return (result == 0);
}
```

Rewrite program such that execution time does not depend on secrets

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

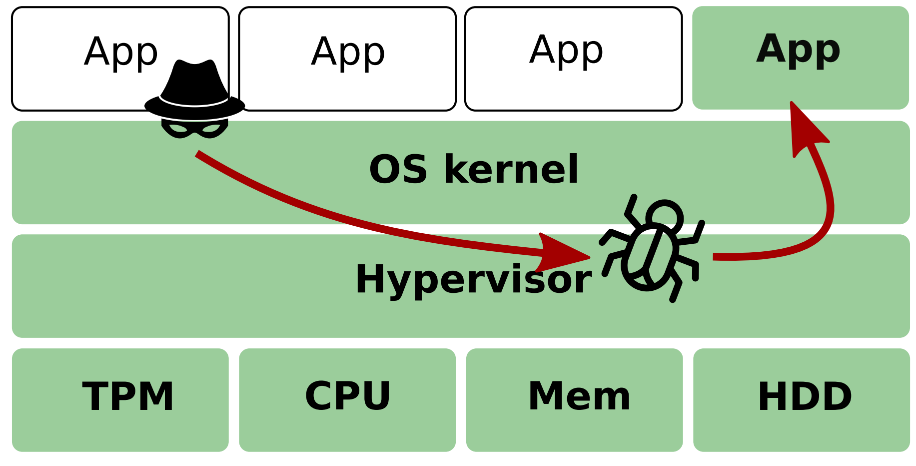


What’s inside the black box?



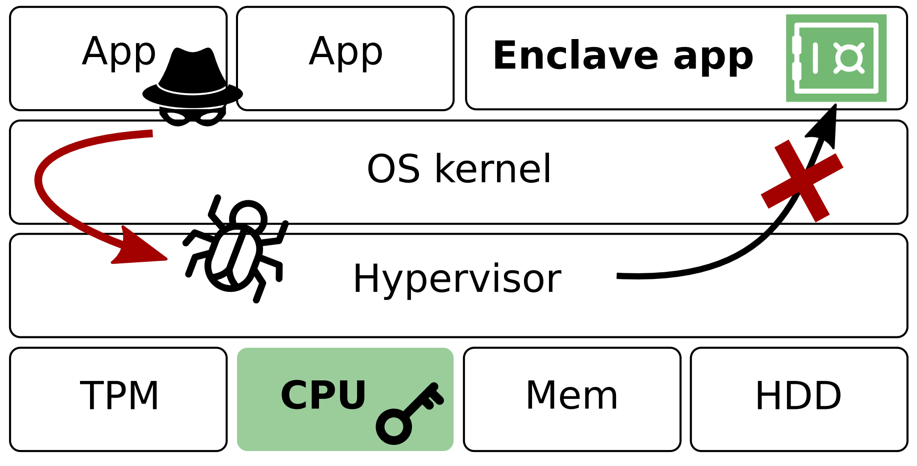
https://informationisbeautiful.net/visualizations/million-lines-of-code/

Enclaved execution: Reducing attack surface



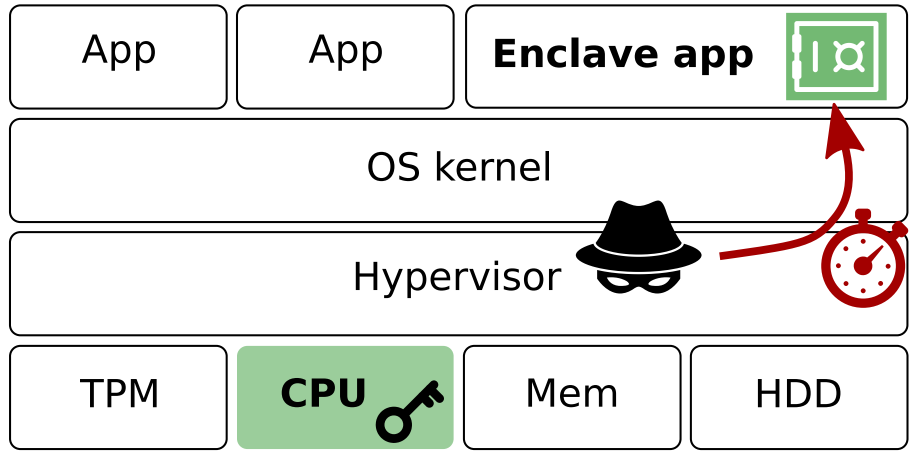
Traditional **layered designs:** large trusted computing base

Enclaved execution: Reducing attack surface



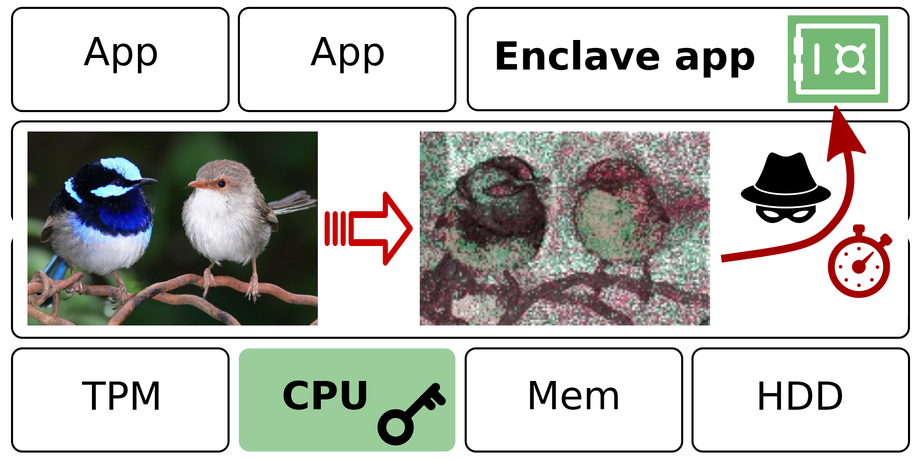
Intel SGX promise: hardware-level **isolation and attestation**

Enclaved execution: Privileged side-channel attacks



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

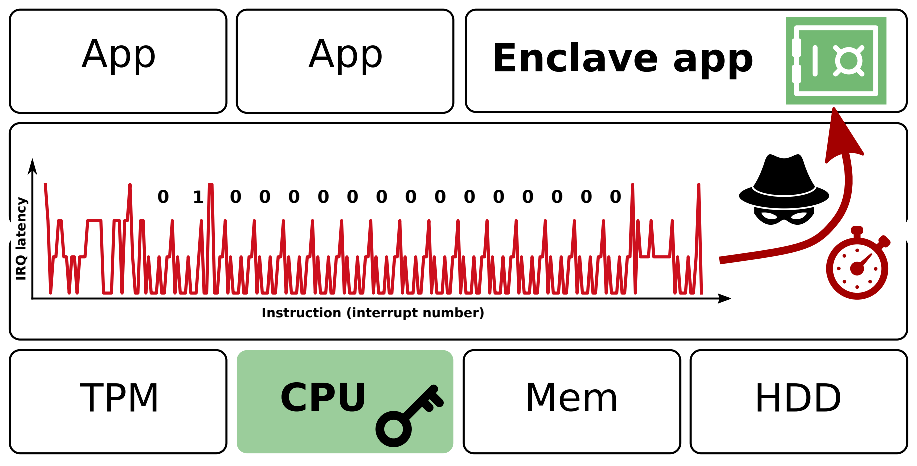
Enclaved execution: Privileged side-channel attacks



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

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

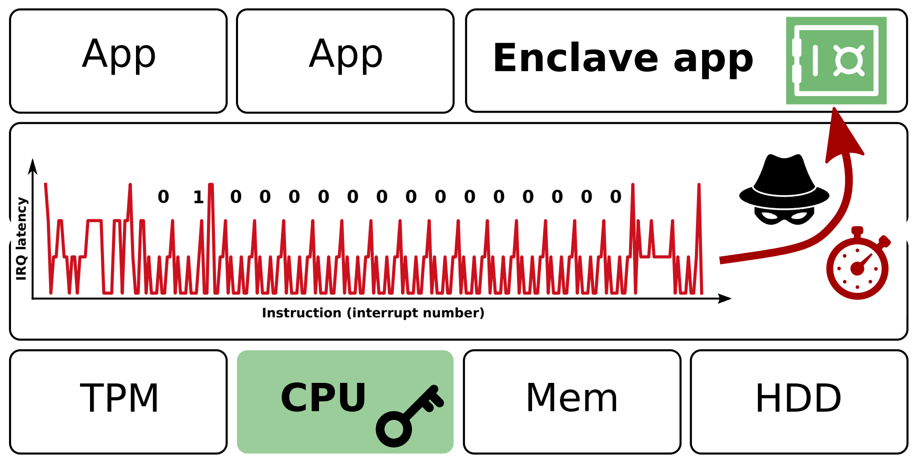
Enclaved execution: Privileged side-channel attacks



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

Van Bulck et al. "Nemesis: Studying Microarchitectural Timing Leaks in Rudimentary CPU Interrupt Logic", CCS 2018

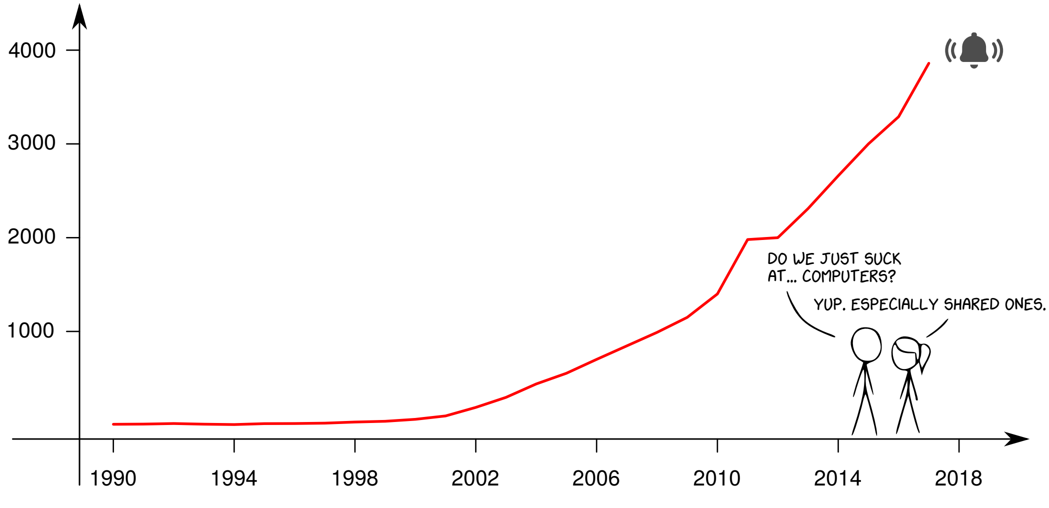
Enclaved execution: Privileged side-channel attacks



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

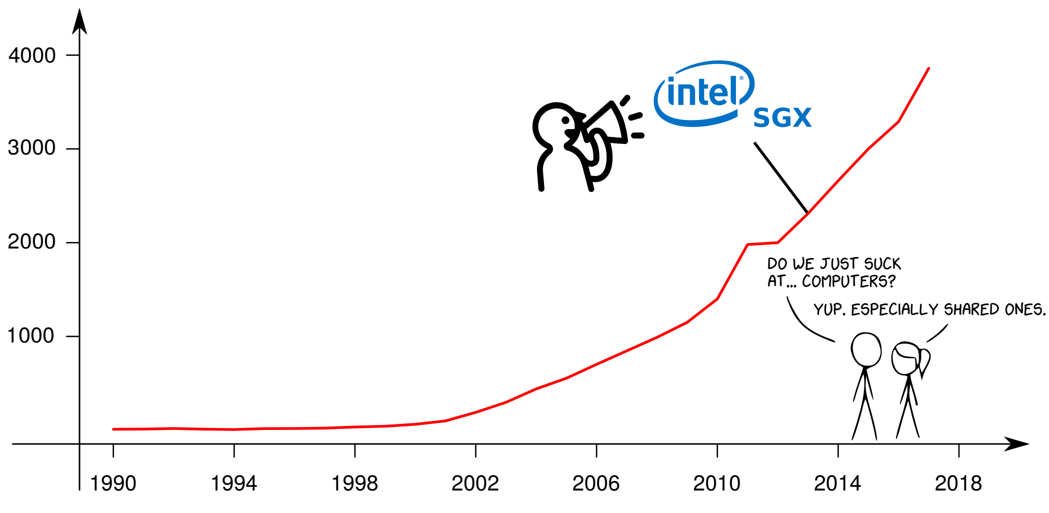
Van Bulck et al. "Nemesis: Studying Microarchitectural Timing Leaks in Rudimentary CPU Interrupt Logic", CCS 2018

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



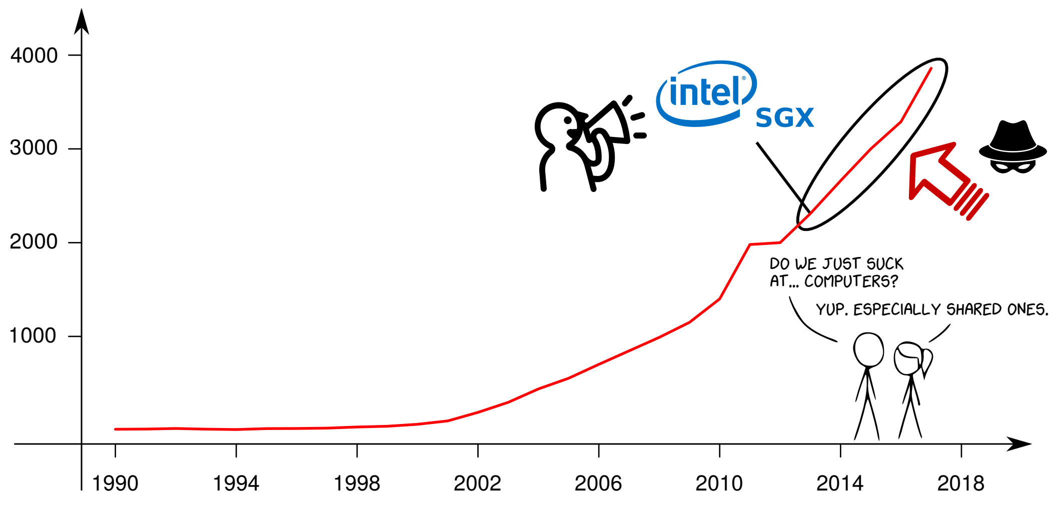
Based on `github.com/Pold87/academic-keyword-occurrence` and `xkcd.com/1938/`

Side-channel attacks and trusted computing

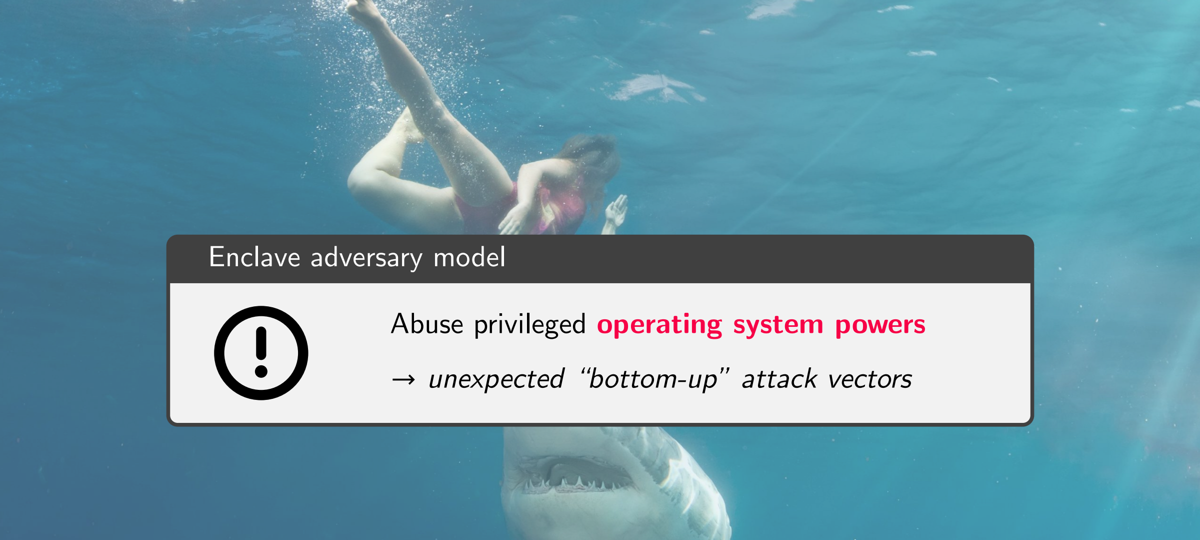


Based on github.com/Pold87/academic-keyword-occurrence and xkcd.com/1938/

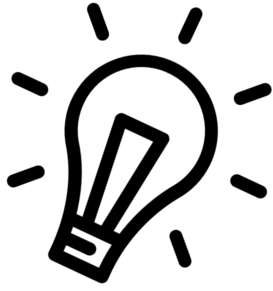
Side-channel attacks and trusted computing (focus of today)



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

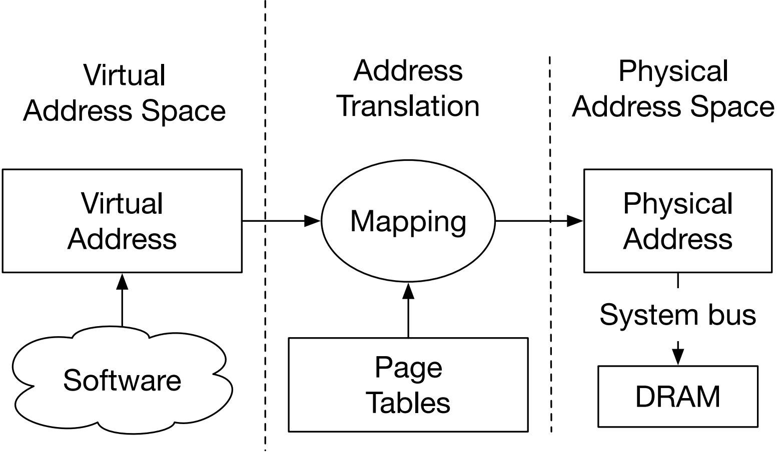

Enclave adversary model
Abuse privileged **operating system powers**
→ *unexpected "bottom-up" attack vectors*

Privileged adversary idea #1



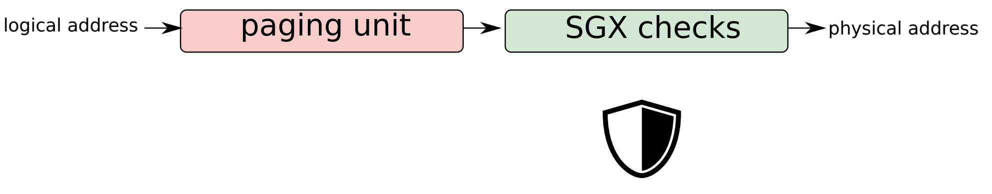
Page tables as a side channel?

The virtual memory abstraction



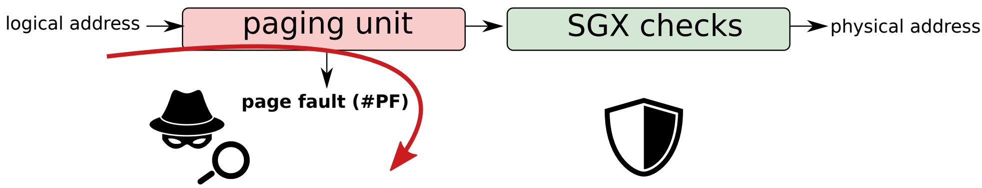
Costan et al. "Intel SGX explained", IACR 2016

Page faults as a side channel



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

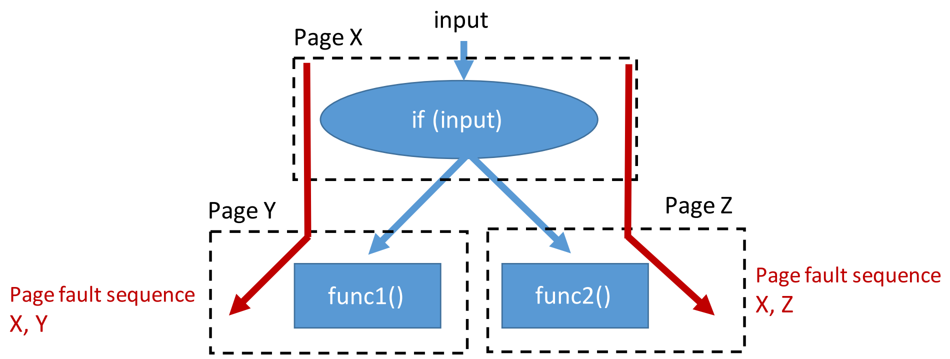
Page faults as a side channel



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

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

Page faults as a side channel



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

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

#PF attacks: An end-to-end example

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

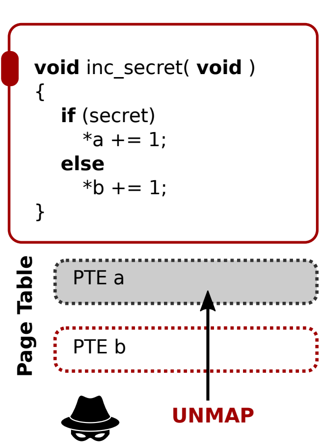
Page Table

PTE a

PTE b

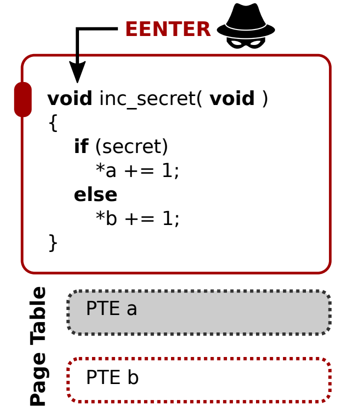
#PF attacks: An end-to-end example

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



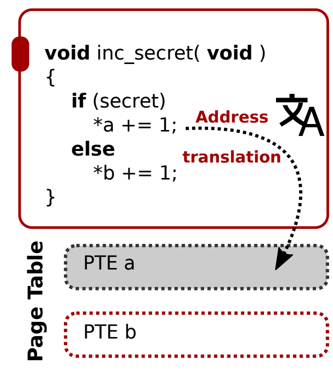
#PF attacks: An end-to-end example

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



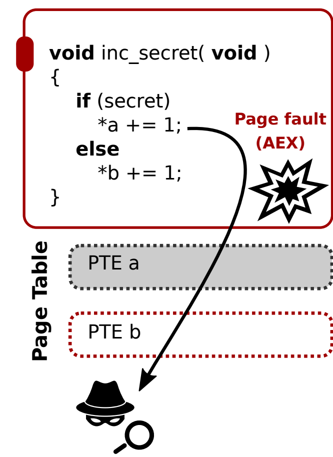
#PF attacks: An end-to-end example

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



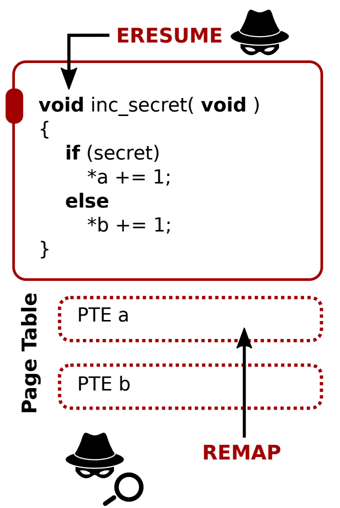
#PF attacks: An end-to-end example

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

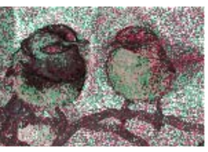
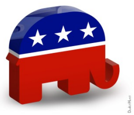


#PF attacks: An end-to-end example

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



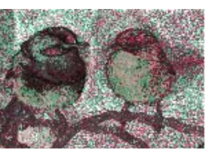
Page table-based attacks in practice

Original	Recovered	Original	Recovered
			
			

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

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

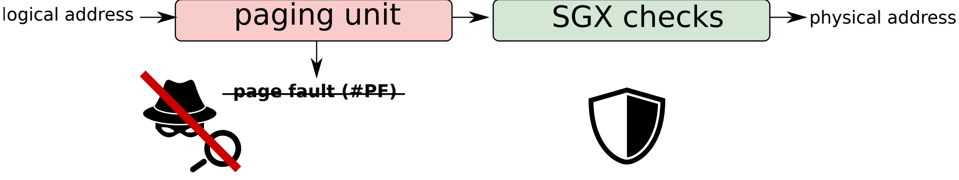
Page table-based attacks in practice

Original	Recovered	Original	Recovered
			

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

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

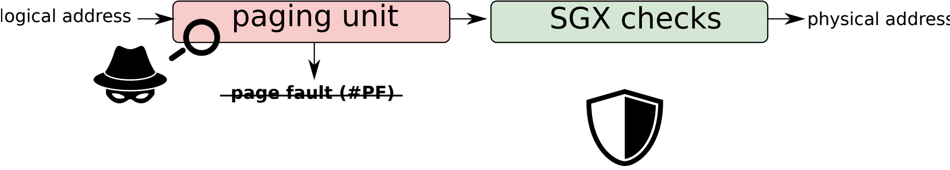
Naive solutions: Hiding enclave page faults



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!

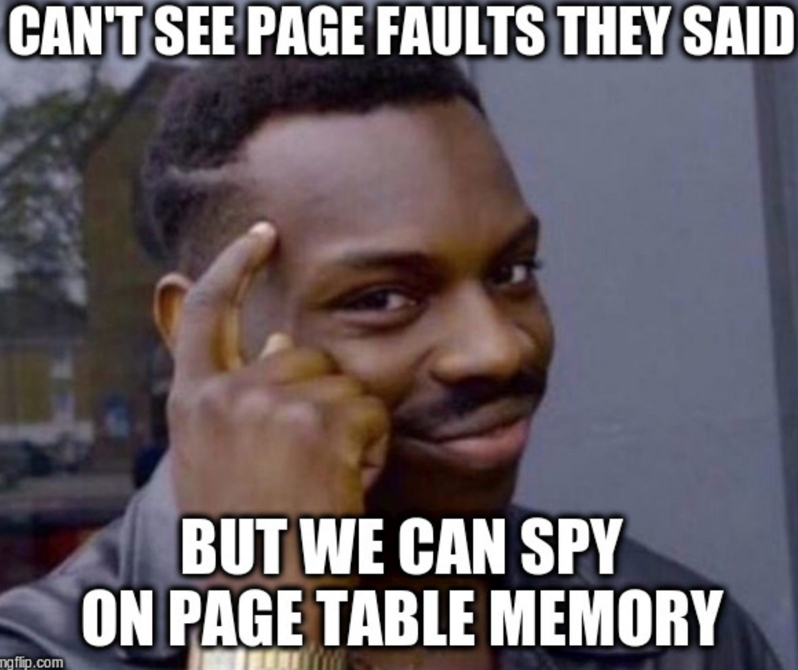
Documented side-effects of address translation

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).

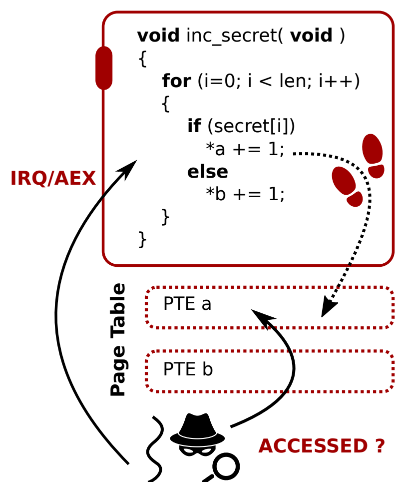
Whenever there is a write to a linear address, the processor sets the dirty flag (if it is not already set) in the paging-structure entry that identifies the final physical address for the linear address (either a PTE or a paging-structure entry in which the PS flag is 1).


CAN'T SEE PAGE FAULTS THEY SAID
BUT WE CAN SPY
ON PAGE TABLE MEMORY
imgflip.com

Telling your secrets without page faults

1. Attack vector: PTE **status flags**:
   - A(ccessed) bit
   - D(irty) bit

   ⇝ Also updated in enclave mode!



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

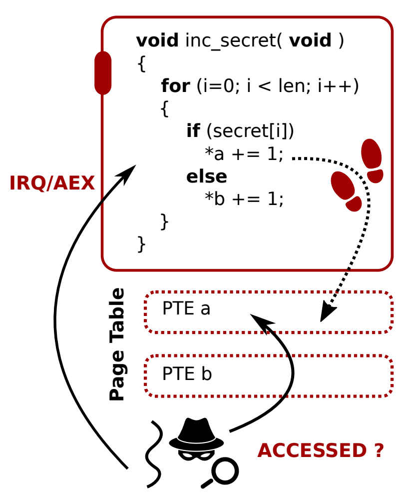
Telling your secrets without page faults

1. Attack vector: PTE **status flags**:
   - A(ccessed) bit
   - D(irty) bit

   ⇝ Also updated in enclave mode!

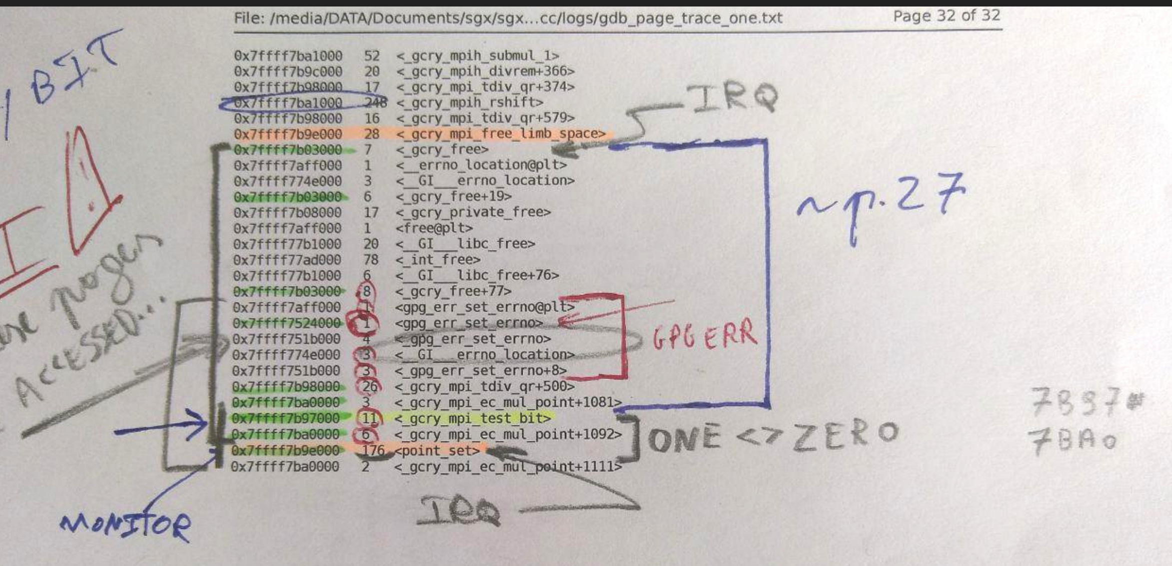
2. Attack vector: Unprotected **page table memory**:
   - Cached as regular data
   - Accessed during address translation

   ⇝ Flush+Reload cache timing attack!



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

Metadata analysis: Page-table access patterns


File: /media/DATA/Documents/sgx/sgx...cc/logs/gdb_page_trace_one.txt


```
0x7ffff7ba1000   52   <_gcry_mpih_submul_1>
0x7ffff7b9c000   20   <_gcry_mpih_divrem+366>
0x7ffff7b98000   17   <_gcry_mpi_tdiv_qr+374>
0x7ffff7ba1000   248  <_gcry_mpih_rshift>
0x7ffff7b98000   16   <_gcry_mpi_tdiv_qr+579>
0x7ffff7b9e000   28   <_gcry_mpi_free_limb_space>
0x7ffff7b03000   7    <_gcry_free>
0x7ffff7aff000   1    <__errno_location@plt>
0x7ffff774e000   3    <__GI___errno_location>
0x7ffff7b03000   6    <_gcry_free+19>
0x7ffff7b08000   17   <_gcry_private_free>
0x7ffff7aff000   1    <free@plt>
0x7ffff77b1000   20   <__GI___libc_free>
0x7ffff77ad000   78   <_int_free>
0x7ffff77b1000   6    <__GI___libc_free+76>
0x7ffff7b03000   8    <_gcry_free+77>
0x7ffff7aff000   1    <gpg_err_set_errno@plt>
0x7ffff7524000   1    <gpg_err_set_errno>
0x7ffff751b000   4    <_gpg_err_set_errno>
0x7ffff774e000   3    <__GI___errno_location>
0x7ffff751b000   3    <_gpg_err_set_errno+8>
0x7ffff7b98000   26   <_gcry_mpi_tdiv_qr+500>
0x7ffff7ba0000   3    <_gcry_mpi_ec_mul_point+1081>
0x7ffff7b97000   11   <_gcry_mpi_test_bit>
0x7ffff7ba0000   6    <_gcry_mpi_ec_mul_point+1092>
0x7ffff7b9e000   176  <point_set>
0x7ffff7ba0000   2    <_gcry_mpi_ec_mul_point+1111>
```

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 }
```


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

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 }
```


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

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 }
```


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

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  }
```


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

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 }
```


Full 512-bit key recovery, single run

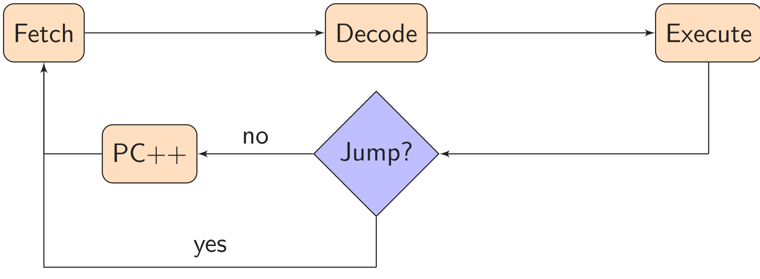
Van Bulck et al. "Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution", USENIX 2017

Privileged adversary idea #2

Interrupts as a side channel?

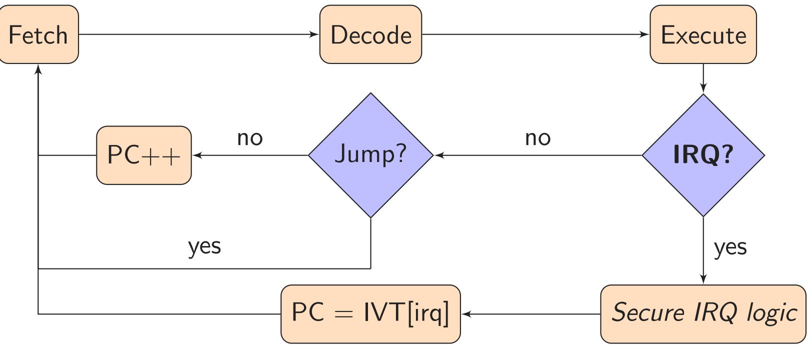
Back to basics: Fetch-decode-execute

Elementary CPU behavior: stored program computer



Back to basics: Fetch-decode-execute

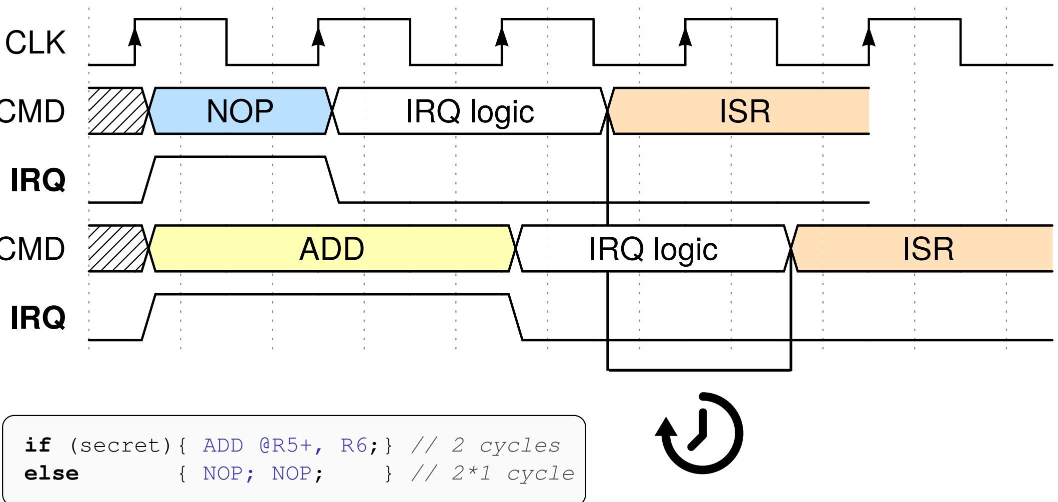
Interrupts: asynchronous real-world events, handled on instruction retirement



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
```


Van Bulck et al. "Nemesis: Studying Microarchitectural Timing Leaks in Rudimentary CPU Interrupt Logic", CCS 2018

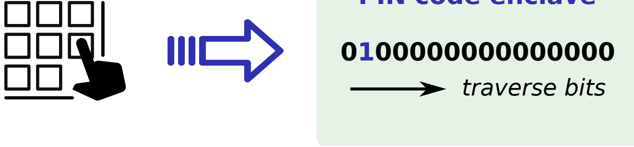
TIMING LEAKS
EVERYWHERE
imgflip.com

Attacking a Sancus application with interrupt latency

Van Bulck et al. “VulCAN: Vehicular component authentication and software isolation”, ACSAC 2017

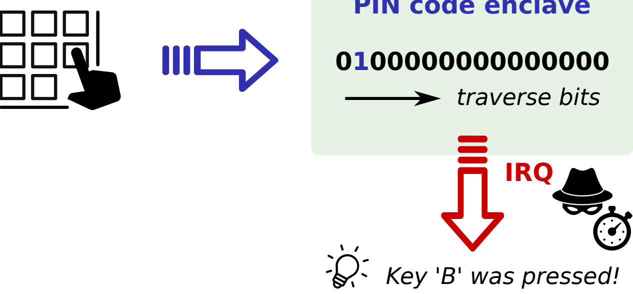
Attacking a Sancus application with interrupt latency

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

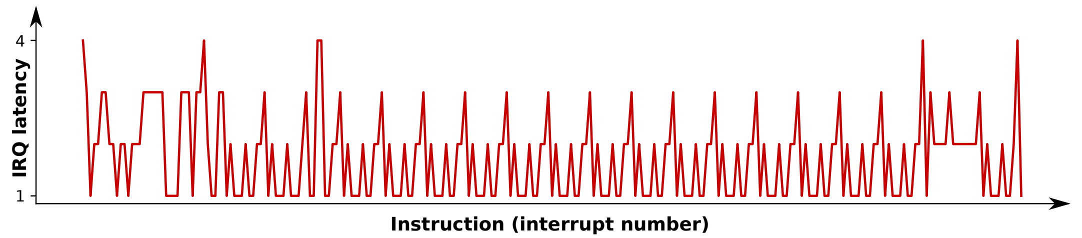


Attacking a Sancus application with interrupt latency

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

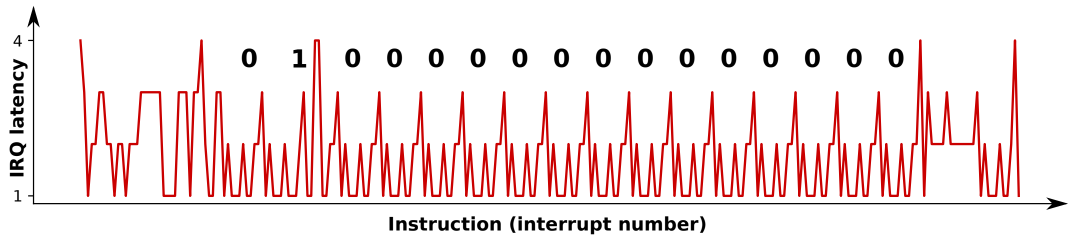


Sancus IRQ timing attack: Inferring key strokes



Van Bulck et al. “Nemesis: Studying Microarchitectural Timing Leaks in Rudimentary CPU Interrupt Logic”, CCS 2018

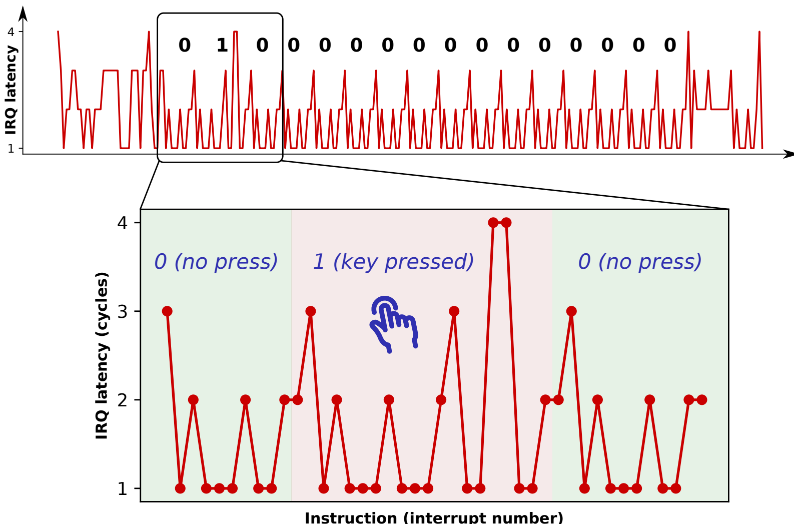
Sancus IRQ timing attack: Inferring key strokes



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

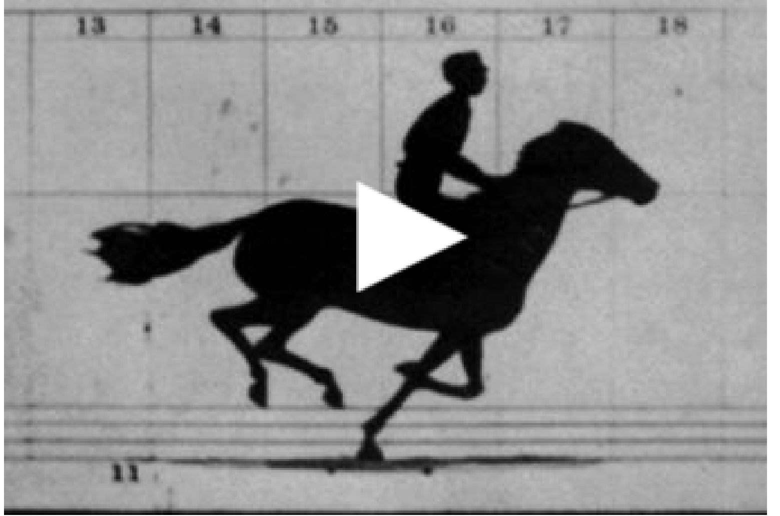
Van Bulck et al. "Nemesis: Studying Microarchitectural Timing Leaks in Rudimentary CPU Interrupt Logic", CCS 2018

Sancus IRQ timing attack: Inferring key strokes

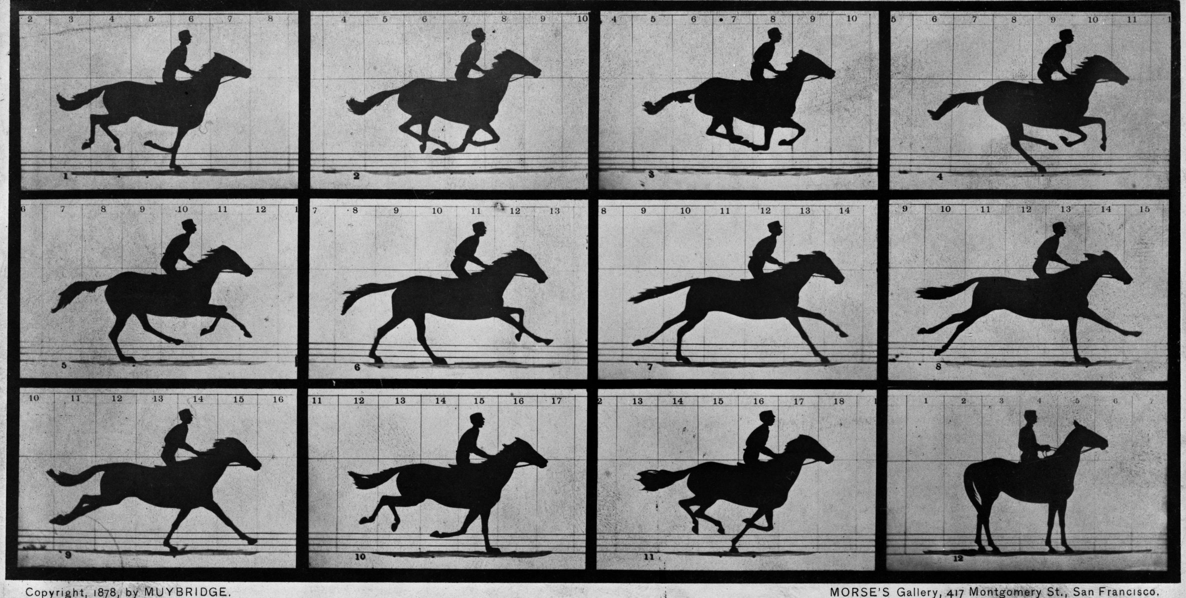


Van Bulck et al. "Nemesis: Studying Microarchitectural Timing Leaks in Rudimentary CPU Interrupt Logic", CCS 2018

Does this also work for Intel SGX enclaves?



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

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!*

Van Bulck et al. "SGX-Step: A practical attack framework for precise enclave execution control", SysTEX 2017

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!*

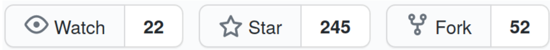
⇒ Setup user-space virtual **memory mappings** for x86 APIC

```
jo@sgx-laptop:~$ cat /proc/iomem | grep "Local APIC"
fee00000-fee00fff : 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:~$
```


Van Bulck et al. "SGX-Step: A practical attack framework for precise enclave execution control", SysTEX 2017

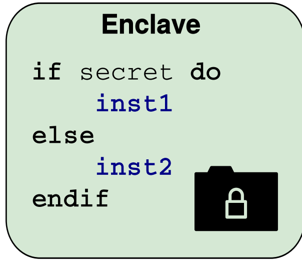
SGX-Step: Executing enclaves one instruction at a time

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



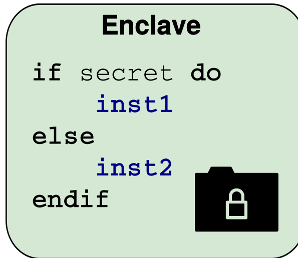
Van Bulck et al. "SGX-Step: A practical attack framework for precise enclave execution control", SysTEX 2017

SGX-Step: Executing enclaves one instruction at a time



Van Bulck et al. "SGX-Step: A practical attack framework for precise enclave execution control", SysTEX 2017

SGX-Step: Executing enclaves one instruction at a time



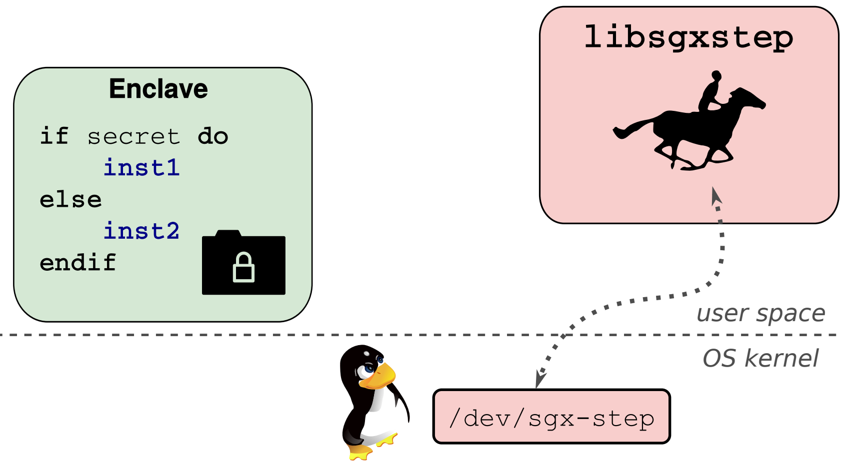
user space

OS kernel

/dev/sgx-step

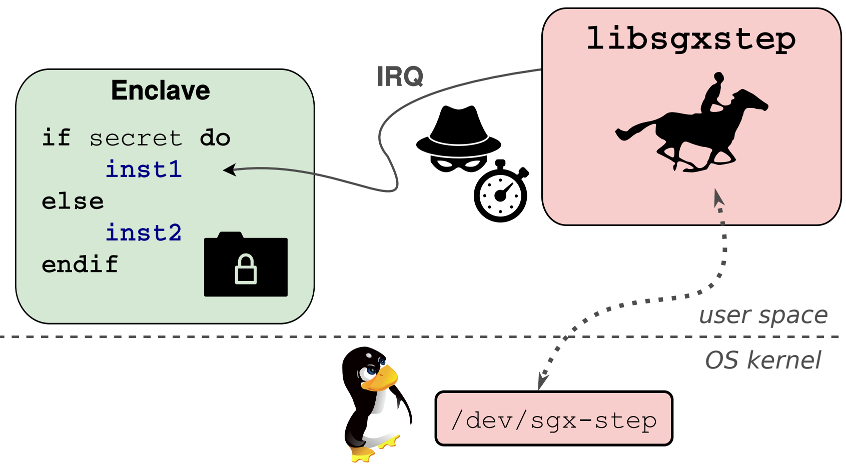
Van Bulck et al. "SGX-Step: A practical attack framework for precise enclave execution control", SysTEX 2017

SGX-Step: Executing enclaves one instruction at a time



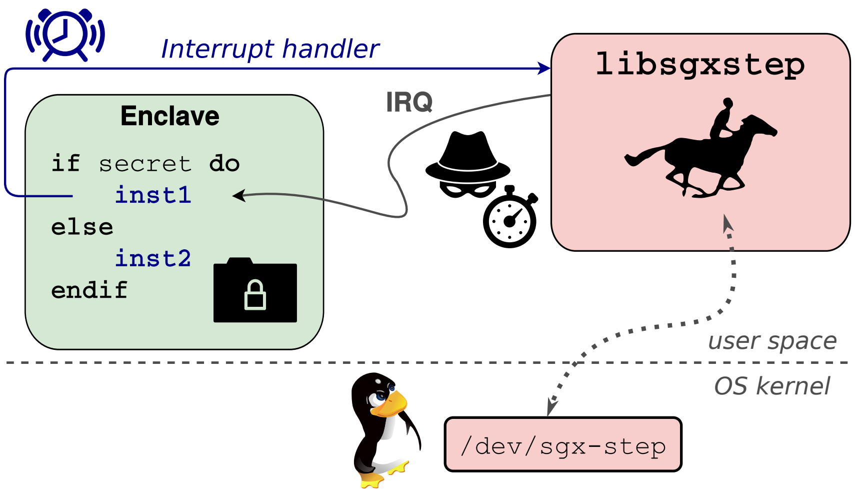
Van Bulck et al. "SGX-Step: A practical attack framework for precise enclave execution control", SysTEX 2017

SGX-Step: Executing enclaves one instruction at a time



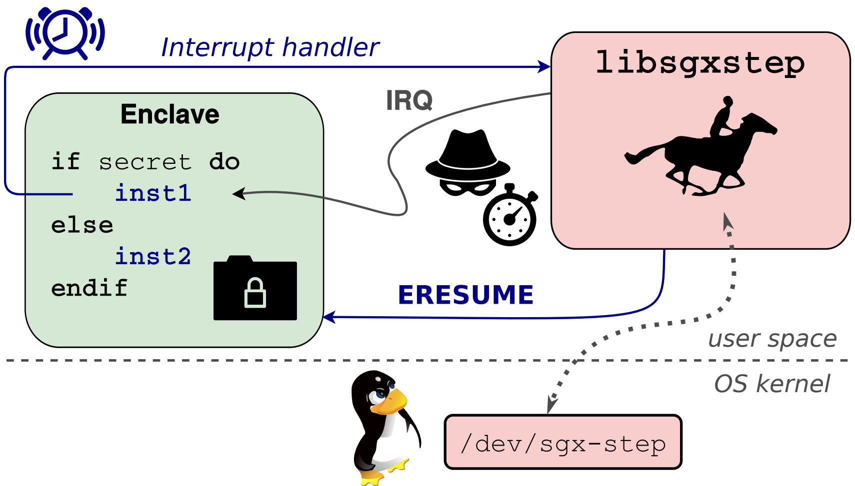
Van Bulck et al. "SGX-Step: A practical attack framework for precise enclave execution control", SysTEX 2017

SGX-Step: Executing enclaves one instruction at a time



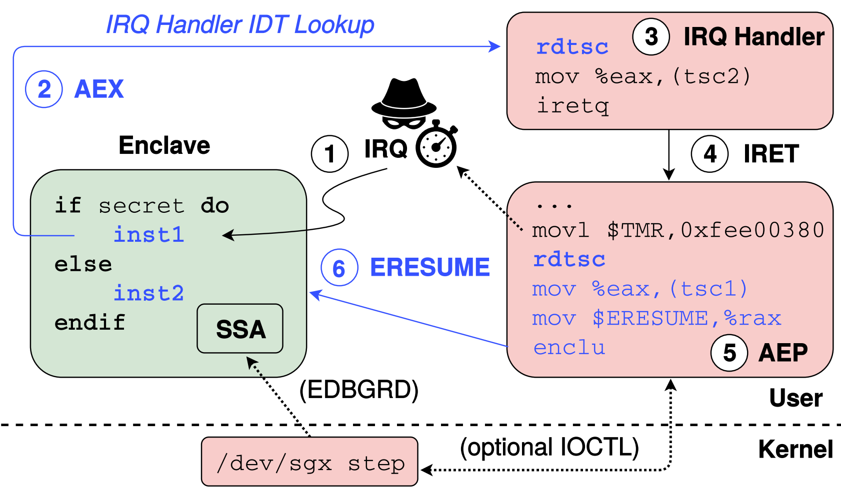
Van Bulck et al. "SGX-Step: A practical attack framework for precise enclave execution control", SysTEX 2017

SGX-Step: Executing enclaves one instruction at a time



Van Bulck et al. "SGX-Step: A practical attack framework for precise enclave execution control", SysTEX 2017

SGX-Step: Executing enclaves one instruction at a time



Van Bulck et al. "SGX-Step: A practical attack framework for precise enclave execution control", SysTEX 2017

Intel SGX Nemesis microbenchmarks: Measuring x86 cache misses

Timing leak: reconstruct *microarchitectural state*

Frequency

load cache hit

load cache miss

7700 7900 8300 8500

IRQ latency (cycles)

Single-stepping Intel SGX enclaves in practice

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

IRQ latency (cycles)

Instruction (interrupt number)

Single-stepping Intel SGX enclaves in practice

Enclave x-ray: Spotting high-latency instructions

rdrand (generate stack canary on enclave entry)

IRQ latency (cycles)

Instruction (interrupt number)

Single-stepping Intel SGX enclaves in practice

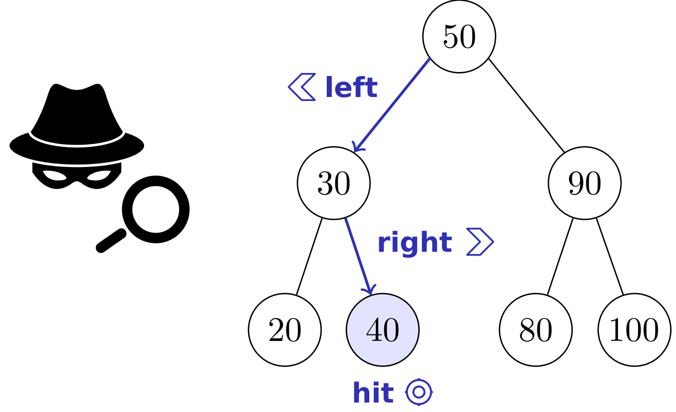
Enclave x-ray: Zooming in on `bsearch` function

IRQ latency (cycles)

Instruction (interrupt number)

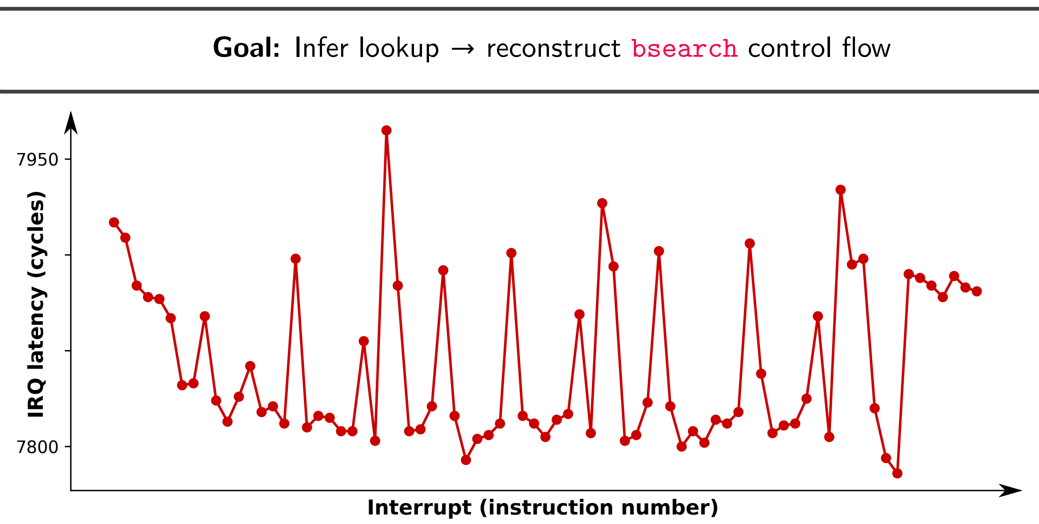
De-anonymizing SGX enclave lookups with interrupt latency

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

7950

7800

IRQ latency (cycles)

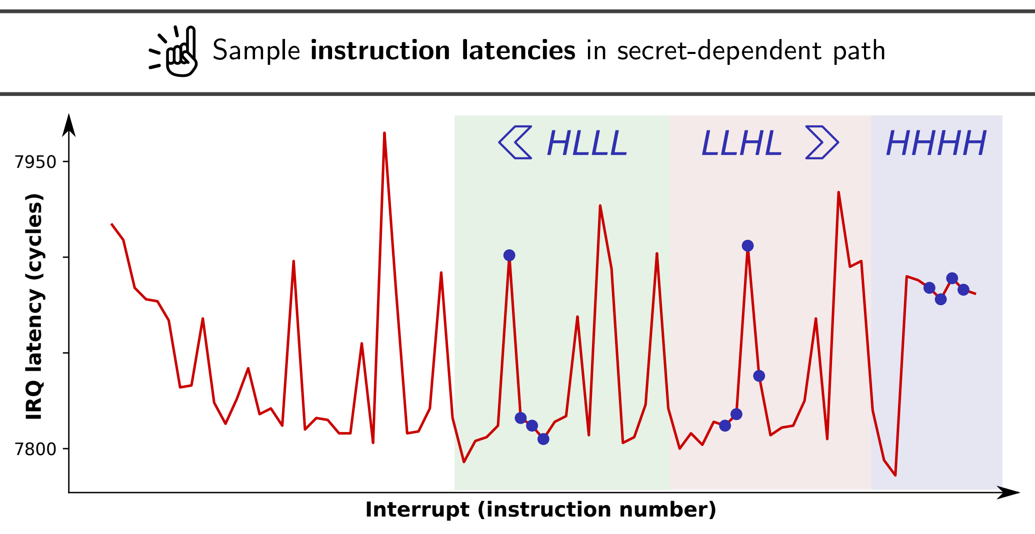
Interrupt (instruction number)

Left

Right

Hit

De-anonymizing SGX enclave lookups with interrupt latency



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

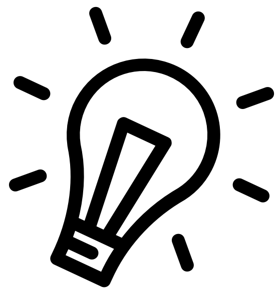
Yr	Attack	Temporal resolution	APIC IRQ	APIC IPI	PTE #PF	PTE A/D	PTE PPN	Desc GDT	Desc IDT	Drv
'15	Ctrl channel	~ Page	○	○	●	○	○	○	●	✓
'16	AsyncShock	~ Page	○	○	●	○	○	○	○	–
'17	CacheZoom	✗ > 1	●	○	○	○	○	○	○	✓
'17	Hahnel et al.	✗ 0 - > 1	●	○	○	○	○	○	●	✓
'17	BranchShadow	✗ 5 - 50	●	○	○	○	○	○	○	✗
'17	Stealthy 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	✗ > 1	●	○	○	○	○	○	○	✓
'18	SGXlinger	✗ > 1	●	○	○	○	○	○	○	✗
'18	Nemesis	✓ 1	●	○	●	●	○	○	●	✓

Yr	Attack	Temporal resolution	APIC IRQ	APIC IPI	PTE #PF	PTE A/D	PTE PPN	Desc GDT	Desc IDT	Drv
'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	○	○	●	○	○	○	○	✓

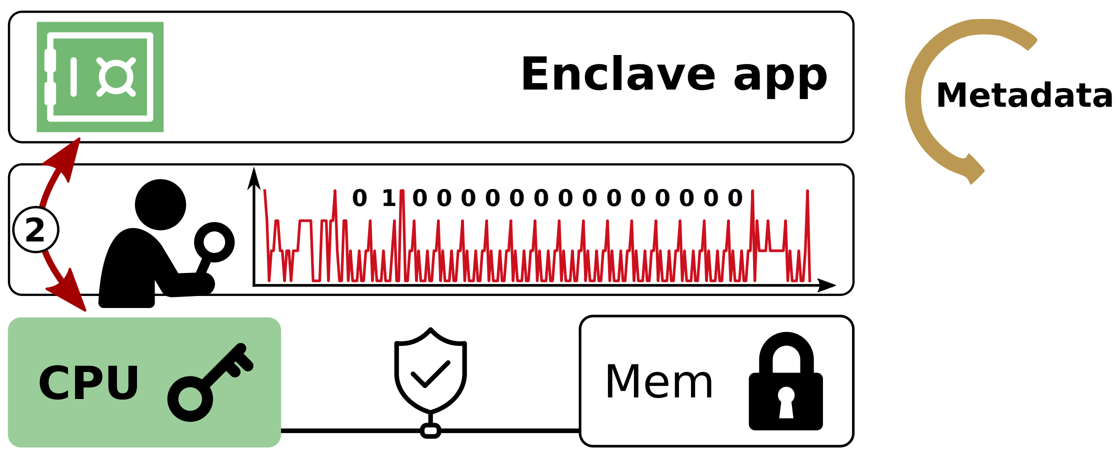
Demo: building a deterministic password oracle with SGX-Step

```
[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/tdcr=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$
```

Privileged adversary idea #3



Page tables revisited: transient execution?


Enclave app
Metadata
2
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CPU
Mem

Outline: Transient-execution attacks

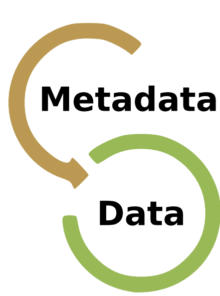
Enclave app

OS kernel

CPU

Mem

3



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.

https://xkcd.com/1938

THE MELTDOWN AND SPECTRE EXPLOITS USE "SPECULATIVE EXECUTION?" WHAT'S THAT?
YOU KNOW THE TROLLEY PROBLEM? WELL,
THE PHANTOM TROLLEY ISN'T SUPPOSED TO TOUCH ANYONE. BUT IT TURNS OUT YOU CAN STILL USE IT TO DO STUFF.
Key finding of 2018
• CPU executes ahead of time in **transient world**
• Use **side channels** to reconstruct secrets!

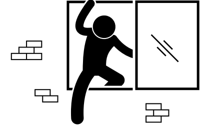
Transient-execution attacks: Welcome to the world of fun!

inside™

inside™

inside™

Meltdown: Transiently encoding unauthorized memory



Unauthorized access

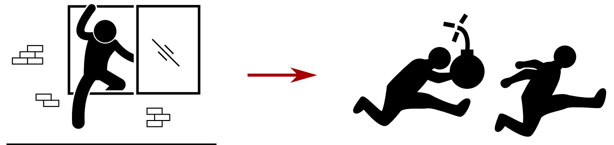
Listing 1: x86 assembly

```
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
```

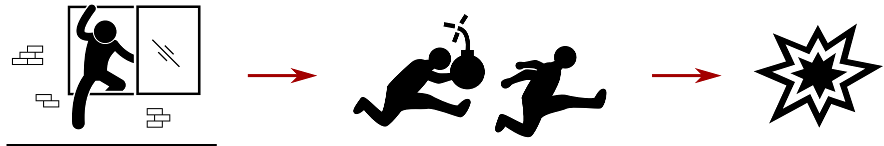
Listing 2: C code.

```
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)

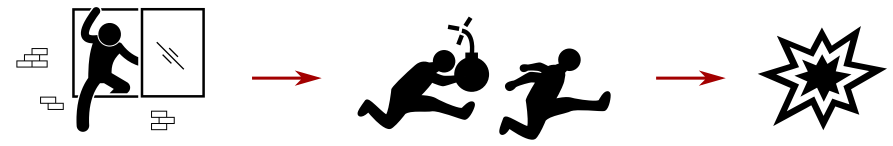
Listing 1: x86 assembly.

```
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

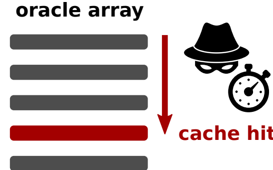


Listing 1: x86 assembly.

```
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 }
```



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

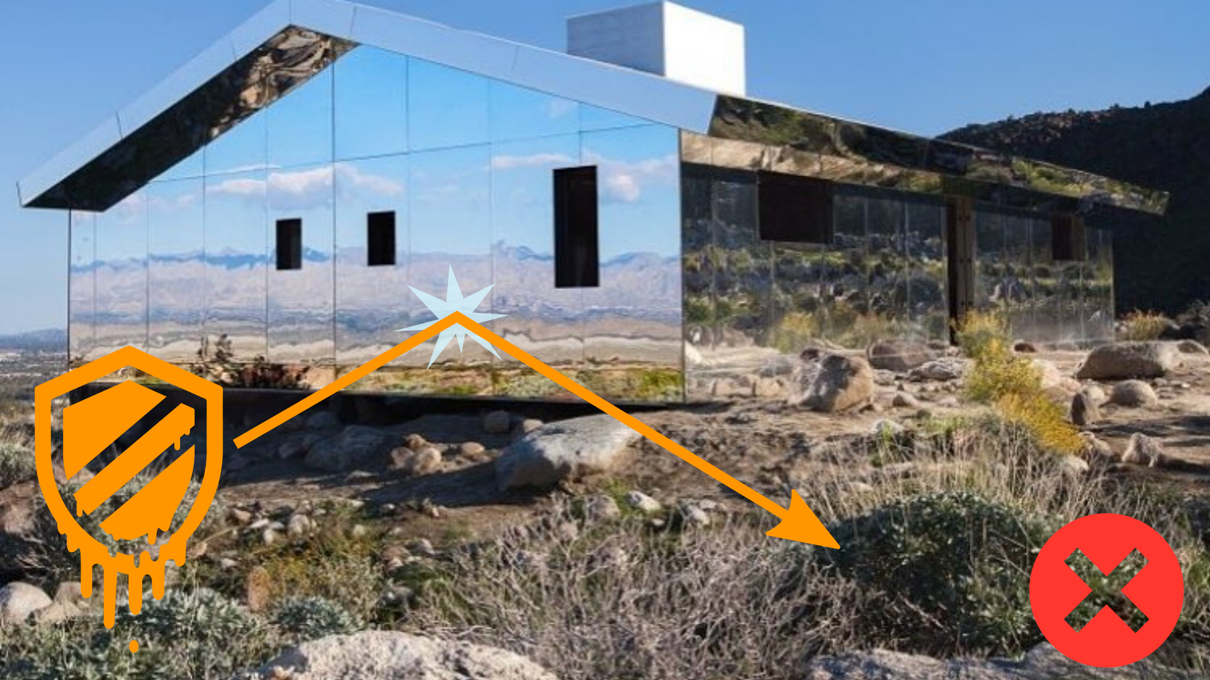
SPECTRE-LIKE FLAW UNDERMINES INTEL PROCESSORS' MOST SECURE ELEMENT

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

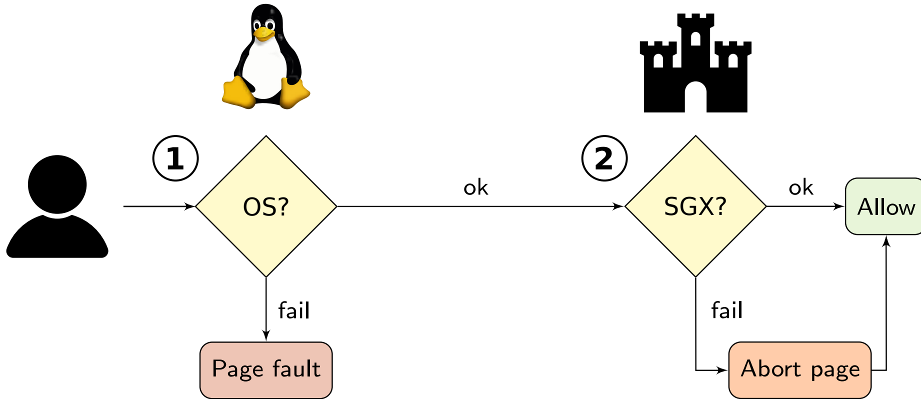
Intel's SGX blown wide open by, you guessed it, a speculative execution attack

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

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

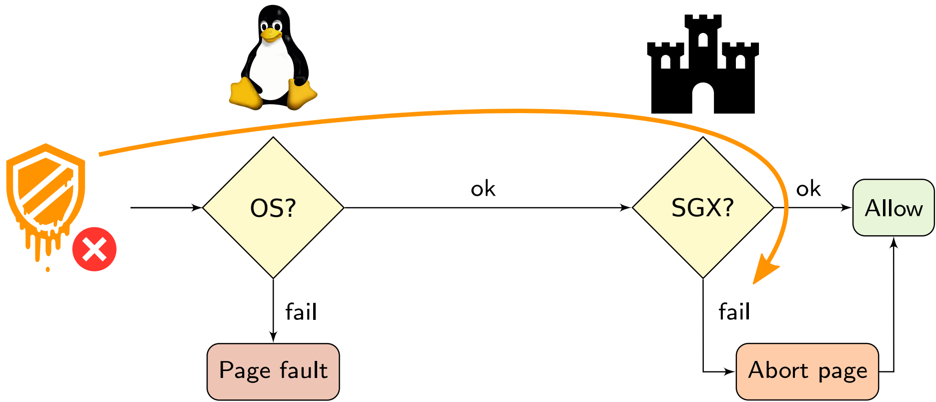


Building Foreshadow: Evade SGX abort page semantics



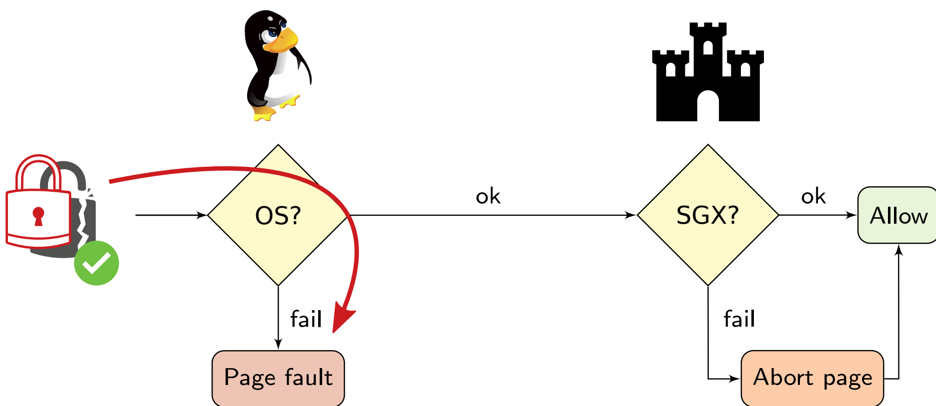
Van Bulck et al. “Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution”, USENIX 2018

Building Foreshadow: Evade SGX abort page semantics



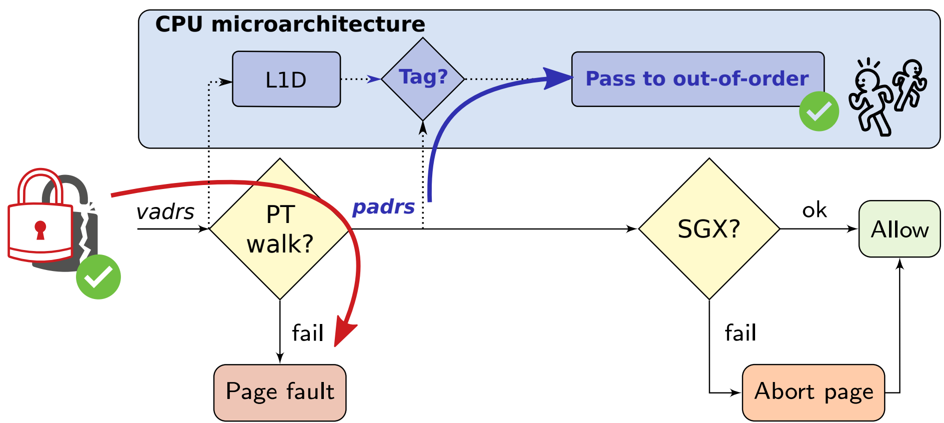
Van Bulck et al. “Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution”, USENIX 2018

Building Foreshadow: Evade SGX abort page semantics



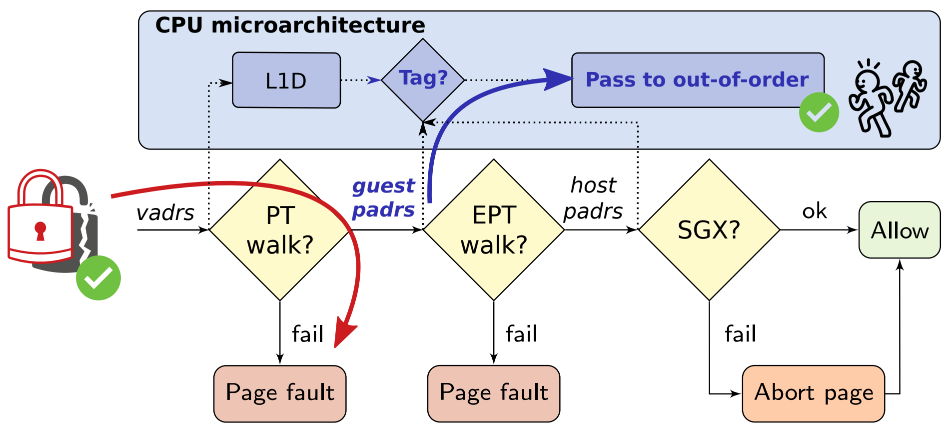
Van Bulck et al. "Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution", USENIX 2018

Foreshadow-SGX: Breaking enclave isolation

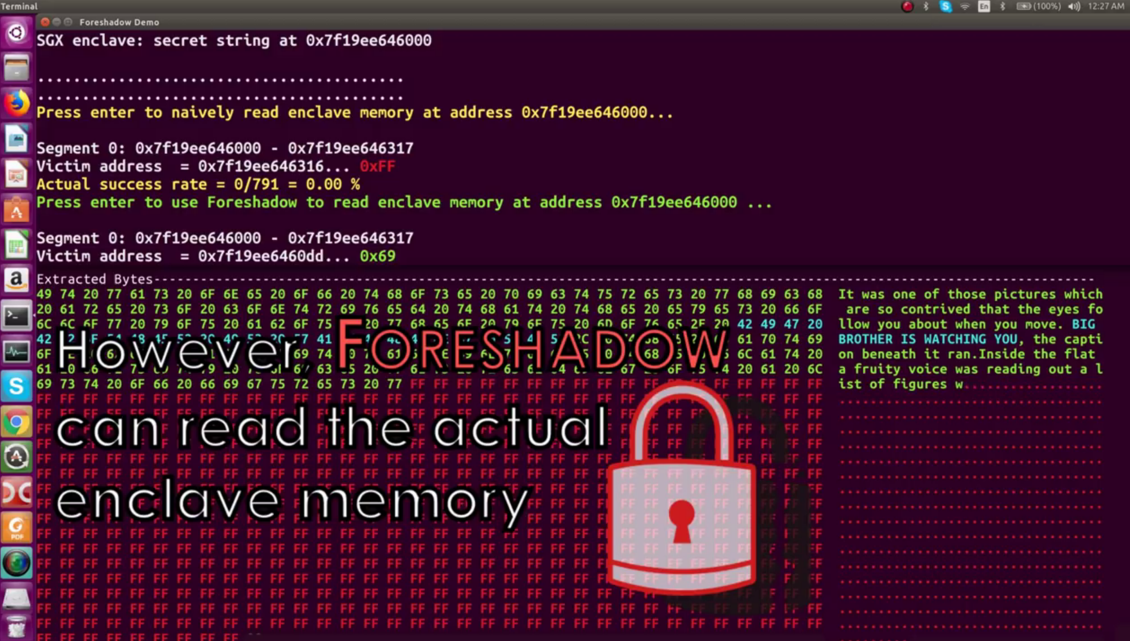


Van Bulck et al. "Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution", USENIX 2018

Foreshadow-NG: Breaking virtual machine isolation



Van Bulck et al. "Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution", USENIX 2018



Terminal
12:27 AM
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 20 6F 6E 65 20 6F 66 20 74 68 6F 73 65 20 70 69 63 74 75 72 65 73 20 77 68 69 63 68
20 61 72 65 20 73 6F 20 63 6F 6E 74 72 69 76 65 64 20 74 68 61 74 20 74 68 65 20 65 79 65 73 20 66 6F
It was one of those pictures which
are so contrived that the eyes fo
llow you about when you move. BIG
BROTHER IS WATCHING YOU, the capti
on beneath it ran.Inside the flat
a fruity voice was reading out a l
ist 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 (WO) 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	

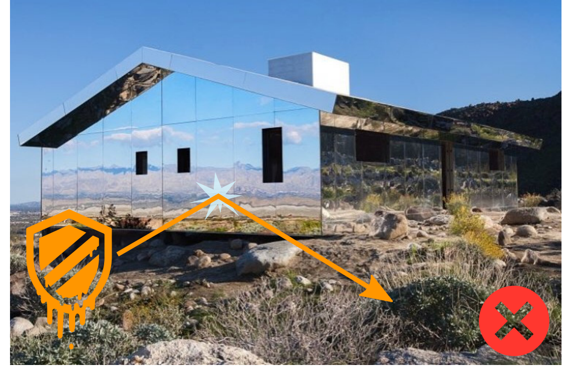
inside™

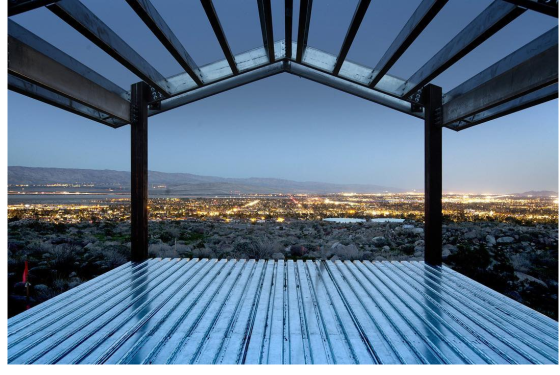
inside™

inside™

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

Idea: Can we turn Foreshadow around?





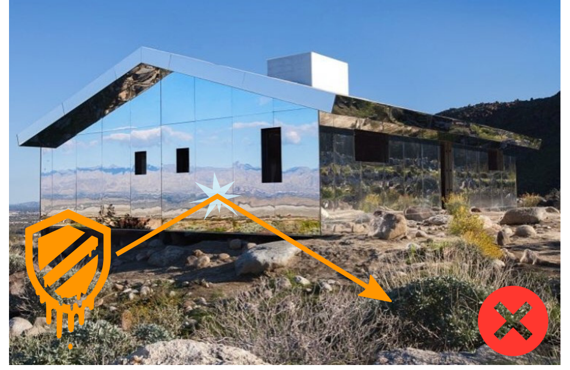
Outside view

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

Intra-enclave view

- Access enclave + outside memory

Idea: Can we turn Foreshadow around?



Outside view

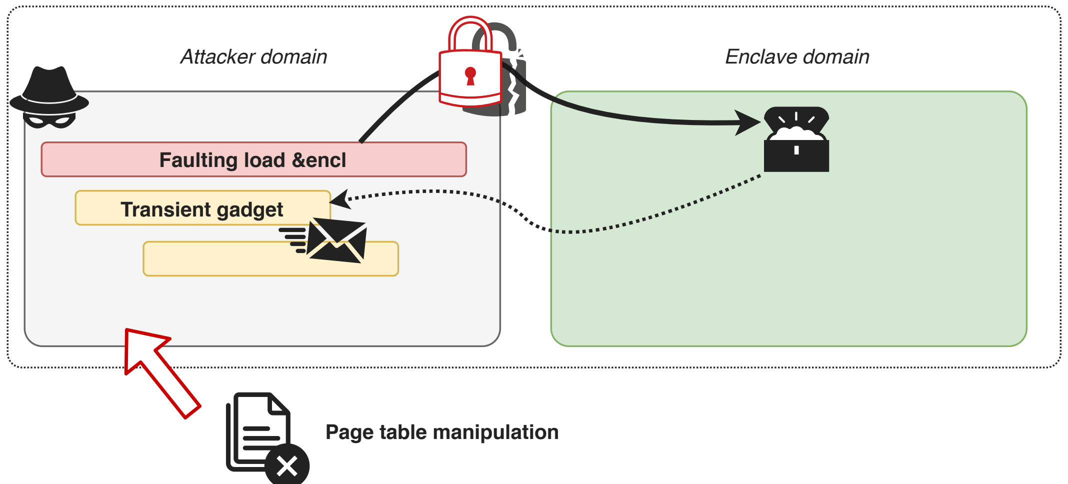
- Meltdown: out-of-reach
- Foreshadow: cache emptied

Intra-enclave view

- Access enclave + outside memory

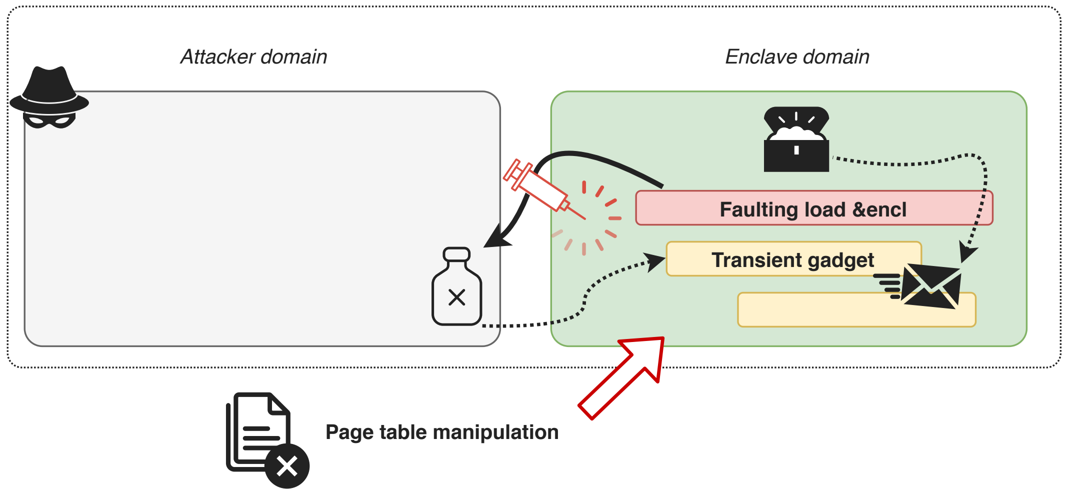
→ Abuse **in-enclave code gadgets!**

Reviving Foreshadow with Load Value Injection (LVI)



Van Bulck et al. “LVI: Hijacking Transient Execution through Microarchitectural Load Value Injection”, S&P 2020.

Reviving Foreshadow with Load Value Injection (LVI)



Van Bulck et al. “LVI: Hijacking Transient Execution through Microarchitectural Load Value Injection”, S&P 2020.

FOOD POISONING

Overdue products

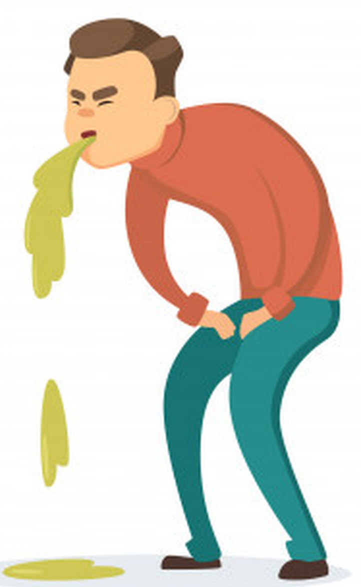
Medicine

Dizziness

Intestinal colic

Diarrhea

Headache



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%

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 0F AE E8	LFENCE	ZO	Valid	Valid	Serializes load operations.

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`

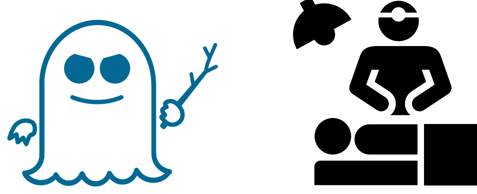
LLVM Lands Performance-Hitting Mitigation For Intel LVI Vulnerability

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

`-Qspectre-load`

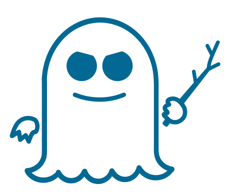
More Spectre Mitigations in

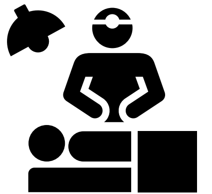
March 13th, 2020

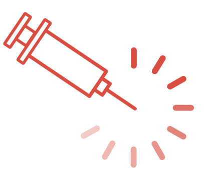


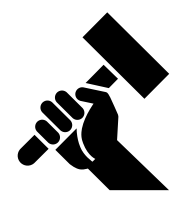
23 fences

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

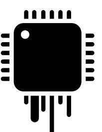
Written by Michael Larabel in Software on 5 April 2020. **Page 1 of 5**. 10 Comments

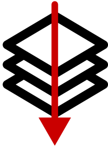
Conclusions and take-away

⇒ **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





Appendix

A note on side-channel attacks (Intel) 2016

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.

https://software.intel.com/en-us/node/703016

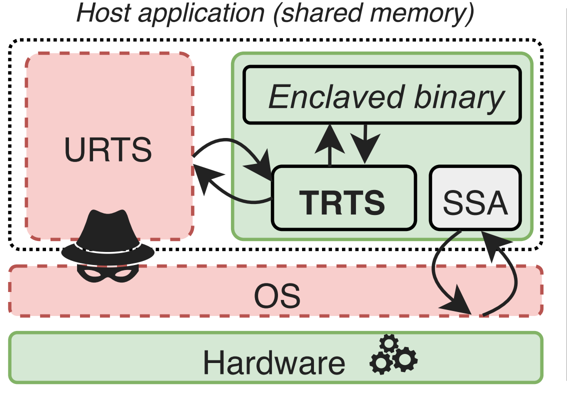
Vulnerable patterns: Secret-dependent code/data accesses

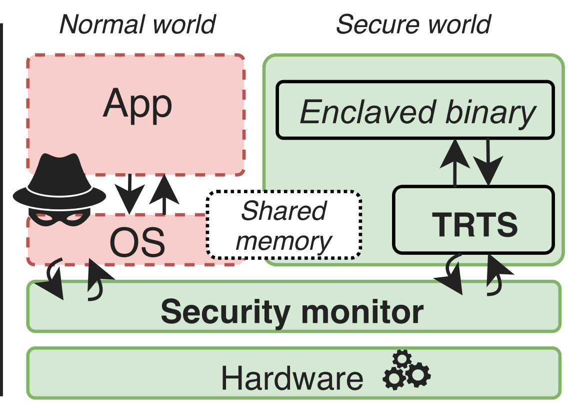
```
1 void secret_vote(char candidate)
2 {
3     if (candidate == 'a')
4         vote_candidate_a();
5     else
6         vote_candidate_b();
7 }
```

```
1 int secret_lookup(int s)
2 {
3     if (s > 0 && s < ARRAY_LEN)
4         return array[s];
5     return -1;
6
7 }
```

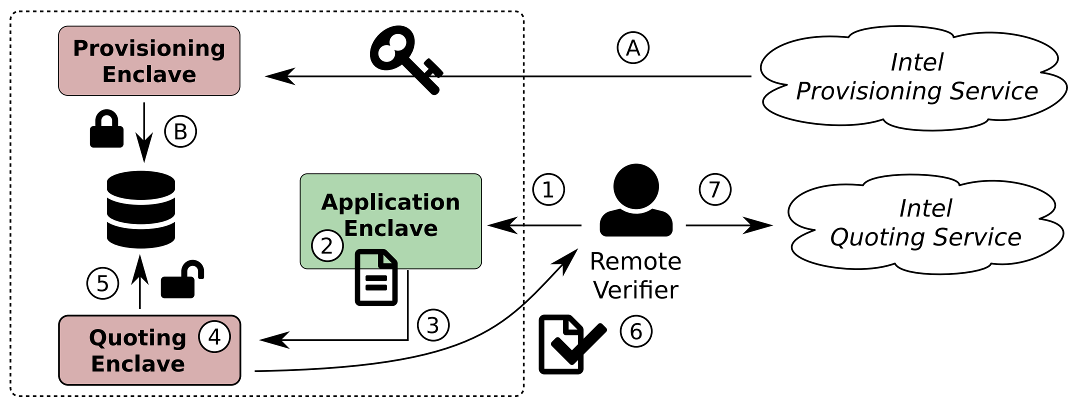
What are the ways for adversaries to create an "oracle" for all victim code+data memory access sequences?

TEE design

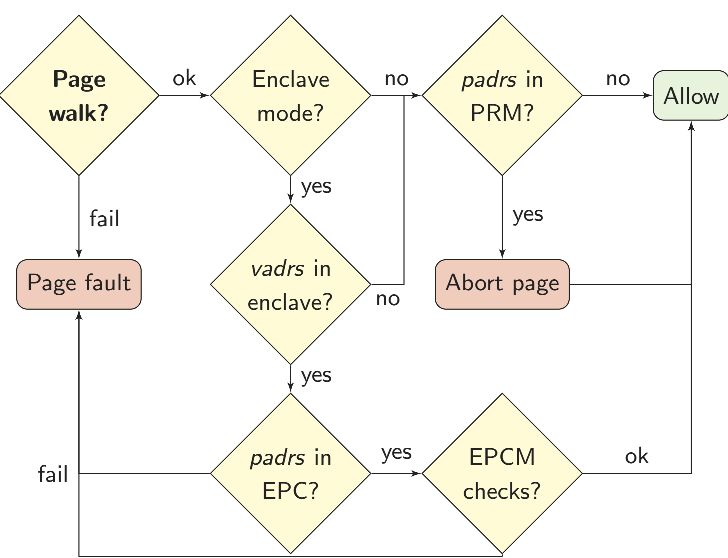




SGX attestation overview



SGX memory access control



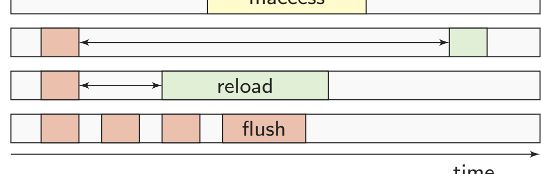
High-resolution, low-latency PTE spying with Flush+Flush

(a) Victim PTE access

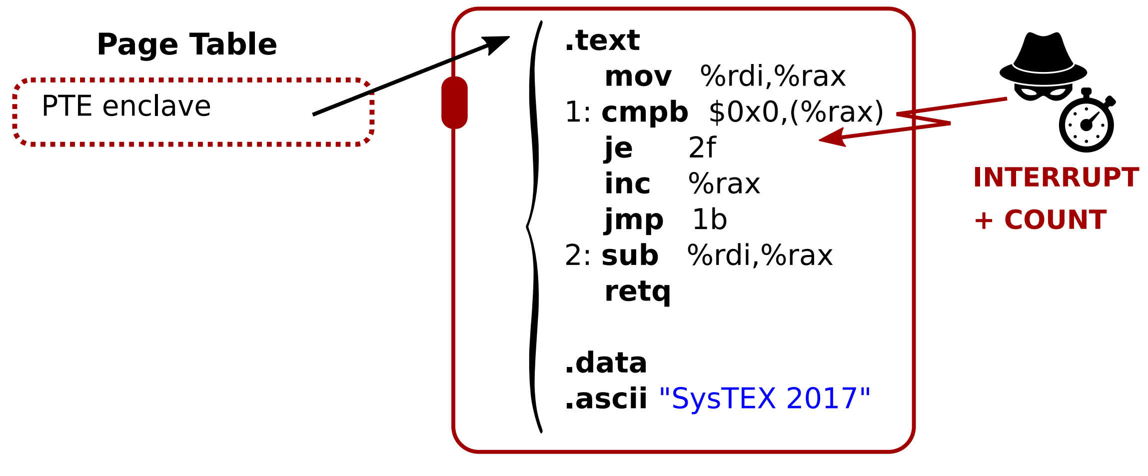
(b) Flush+Reload hit

(c) Flush+Reload miss

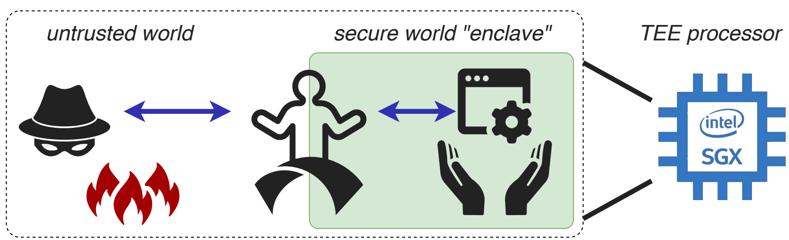
(d) Flush+Flush hit



Improving page-table attack resolution with interrupt counting



Why isolation is not enough: Enclave shielding runtimes

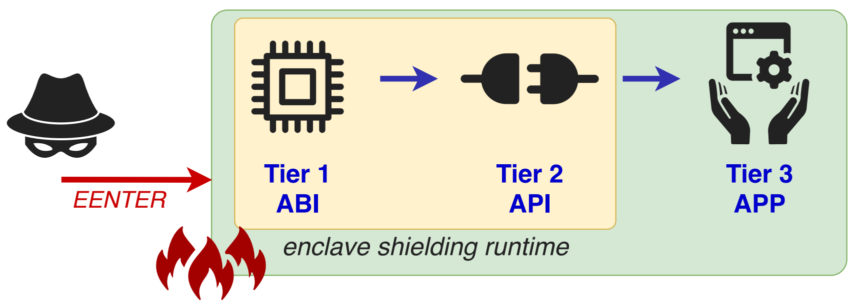


- TEE promise: enclave == "secure oasis" in a **hostile environment**
- ... but app writers and compilers are largely unaware of **isolation boundaries**

Trusted **shielding runtime** transparently acts as a secure bridge on enclave entry/exit

Enclave shielding responsibilities

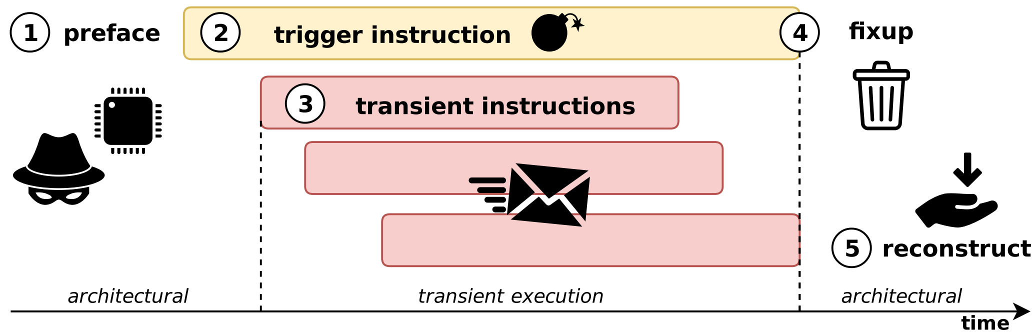
Key insight: split sanitization responsibilities across the ABI and API tiers: *machine state* vs. higher-level *programming language interface*



Tale of 2 worlds vulnerability assessment matrix

	Vulnerability \ Runtime	SGX-SDK	OpenEnclave	Graphene	SGX-LKL	Rust-EDP	Asylo	Keystone	Sancus
ABI	#1 Entry status flags sanitization	★	★	◐	●	◐	●	○	○
	#2 Entry stack pointer restore	○	○	★	●	○	○	○	★
	#3 Exit register leakage	○	○	○	★	○	○	○	○
Tier2 (API)	#4 Missing pointer range check	○	★	★	★	○	●	○	★
	#5 Null-terminated string handling	⯪	★	○	○	○	○	○	○
	#6 Integer overflow in range check	○	○	●	○	●	○	●	●
	#7 Incorrect pointer range check	○	○	●	○	○	●	○	●
	#8 Double fetch untrusted pointer	○	○	●	○	○	○	○	○
	#9 Ocall return value not checked	○	★	★	★	○	●	★	○
	#10 Uninitialized padding leakage	[Lee17]	★	○	●	○	●	★	★

Transient-execution attack overview



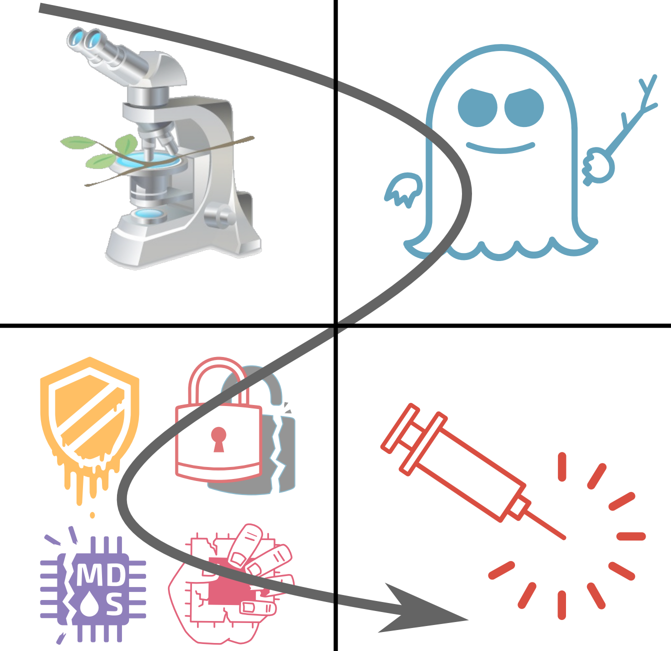
Spectre v1: Speculative buffer over-read

LEN

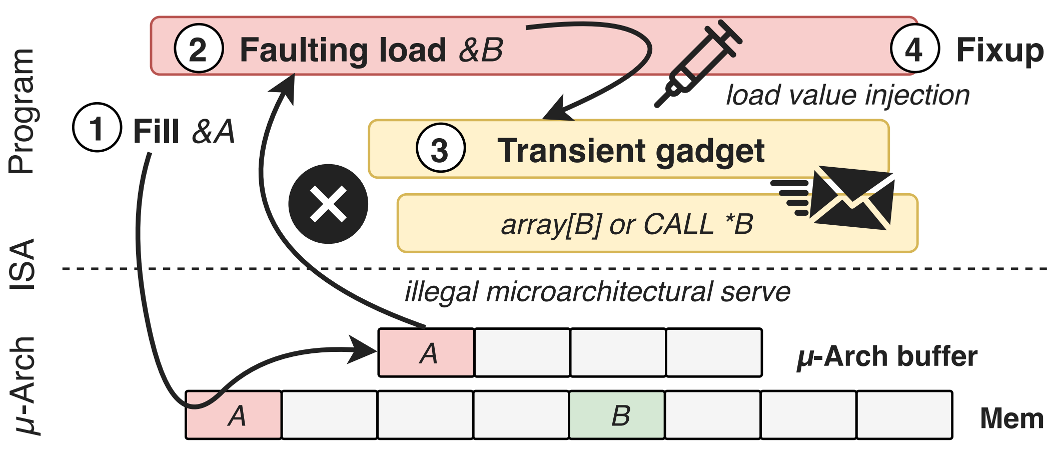
user buffer	*secret*

```
if (idx < LEN)
{
  asm("lfence\n\t");
  s = buffer[idx];
  t = lookup[s];
  ...
}
```

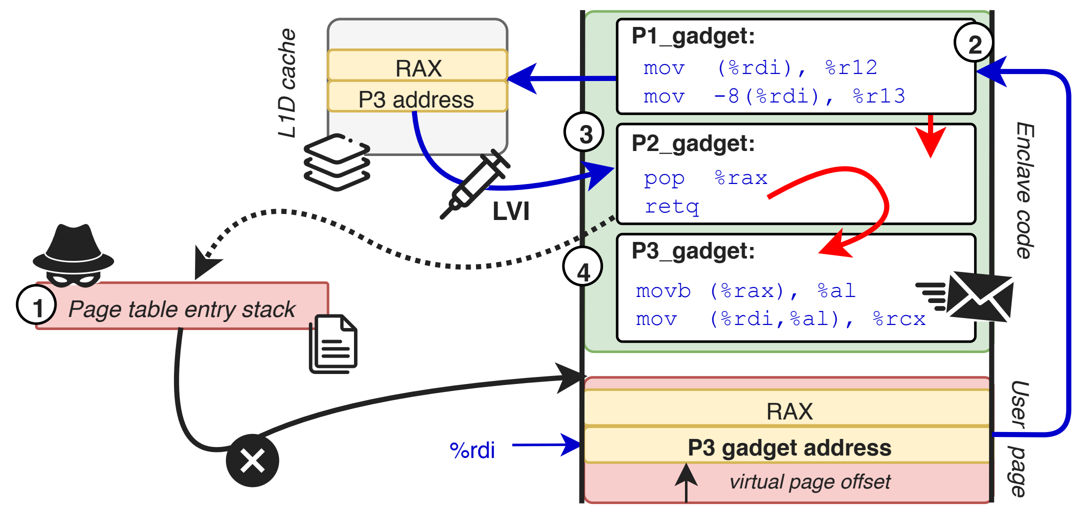
- Programmer *intention:* never access out-of-bounds
- Branch can be mistrained to speculatively (i.e., ahead of time) execute with $idx \geq LEN$ in the **transient world**
- Insert explicit **speculation barriers** to tell the CPU to halt the transient world...
- Huge manual, error-prone effort...


MD
S

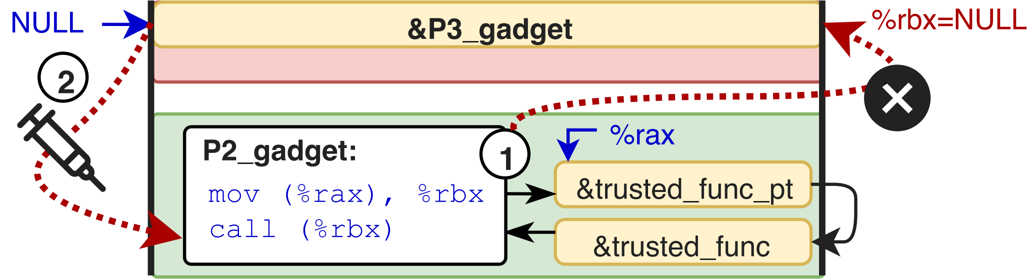
LVI overview



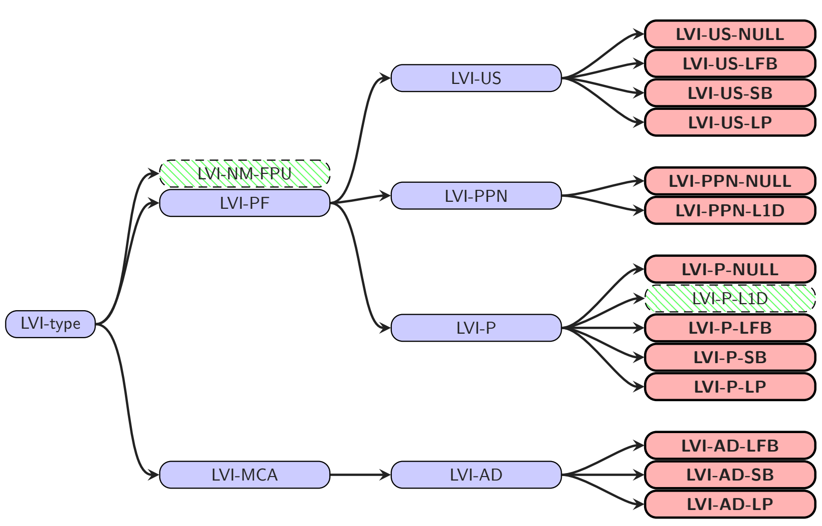
LVI-based control-flow hijacking



LVI-NULL function-pointer hijacking



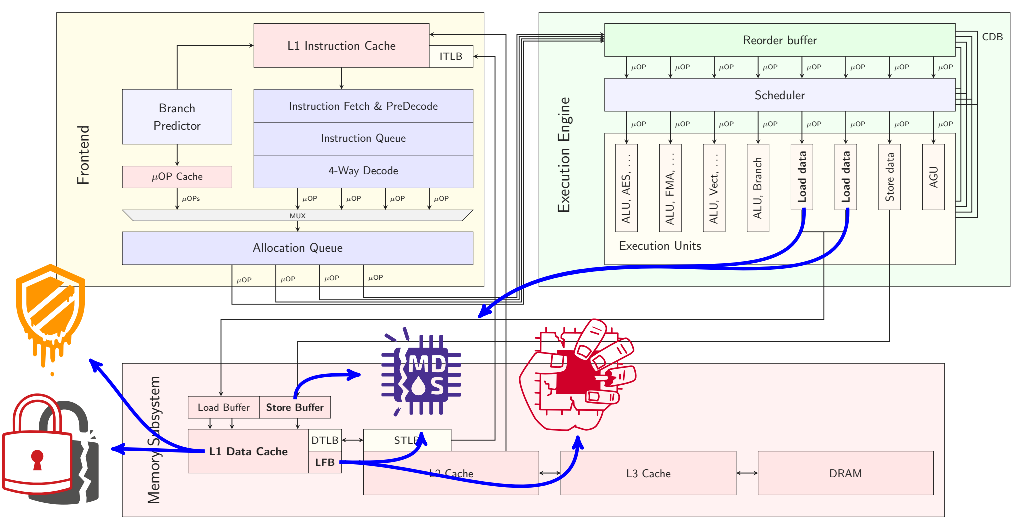
Taxonomy of LVI variants



Meltdown variants: Address dependencies

	Page Number	Page Offset
Meltdown	51 Physical 12 47 Virtual 12	11 0
Foreshadow	51 Physical 12 47 Virtual 12	11 0
Fallout	51 Physical 12 47 Virtual 12	11 0
ZombieLoad/ RIDL	51 Physical 12 47 Virtual 12	11 6 5 0

Meltdown variants: Microarchitectural buffers



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.
In *28th USENIX Security Symposium*, pp. 249–266, August 2019.

D. Moghimi, J. Van Bulck, N. Heninger, F. Piessens, and B. Sunar.
CopyCat: Controlled instruction-level attacks on enclaves.
In *29th USENIX Security Symposium*, pp. 469–486, August 2020.

J. Van Bulck, J. T. Mühlberg, and F. Piessens.
VulCAN: Efficient component authentication and software isolation for automotive control networks.
In *33rd Annual Computer Security Applications Conference (ACSAC)*, pp. 225–237, December 2017.

J. Van Bulck, D. Moghimi, M. Schwarz, M. Lipp, M. Minkin, D. Genkin, Y. Yuval, B. Sunar, D. Gruss, and F. Piessens.
LVI: Hijacking transient execution through microarchitectural load value injection.
In *41st IEEE Symposium on Security and Privacy (S&P)*, pp. 54–72, May 2020.

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 *27th USENIX Security Symposium*, pp. 991–1008, August 2018.

J. Van Bulck, D. Oswald, E. Marin, A. Aldoseri, F. D. Garcia, and F. Piessens.
A tale of two worlds: Assessing the vulnerability of enclave shielding runtimes.
In *26th ACM Conference on Computer and Communications Security (CCS)*, pp. 1741–1758, November 2019.

J. Van Bulck, F. Piessens, and R. Strackx.
SGX-Step: A practical attack framework for precise enclave execution control.
In *2nd Workshop on System Software for Trusted Execution (SysTEX)*, pp. 4:1–4:6. ACM, October 2017.

J. Van Bulck, F. Piessens, and R. Strackx.
Nemesis: Studying microarchitectural timing leaks in rudimentary CPU interrupt logic.
In *25th ACM Conference on Computer and Communications Security (CCS)*, pp. 178–195, October 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 *26th USENIX Security Symposium*, pp. 1041–1056, August 2017.

Y. Xu, W. Cui, and M. Peinado.
Controlled-channel attacks: Deterministic side channels for untrusted operating systems.
In *36th IEEE Symposium on Security and Privacy (S&P)*, pp. 640–656, 2015.