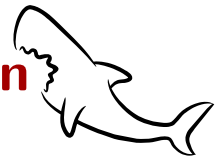
Privileged Side-Channel Attacks on Trusted Execution Environments



Jo Van Bulck

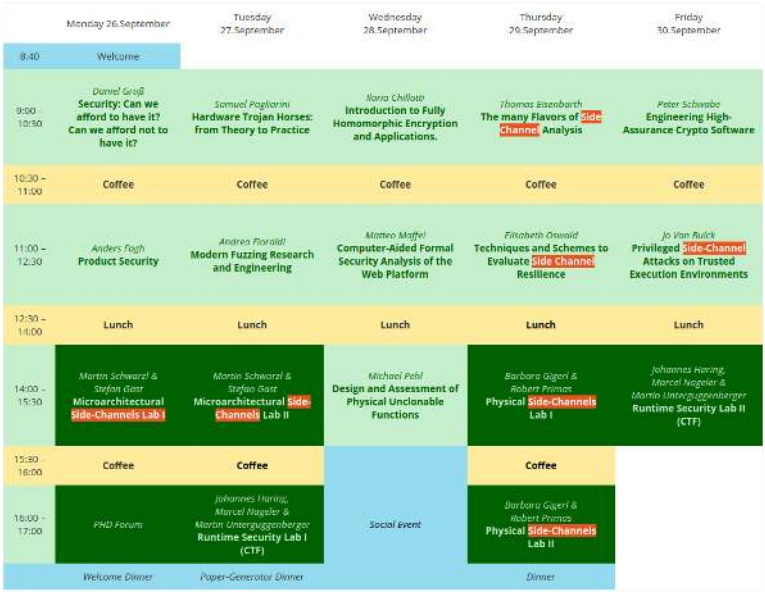
Summer School on Security & Correctness, TU Graz, September 30, 2022

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

DistriNet

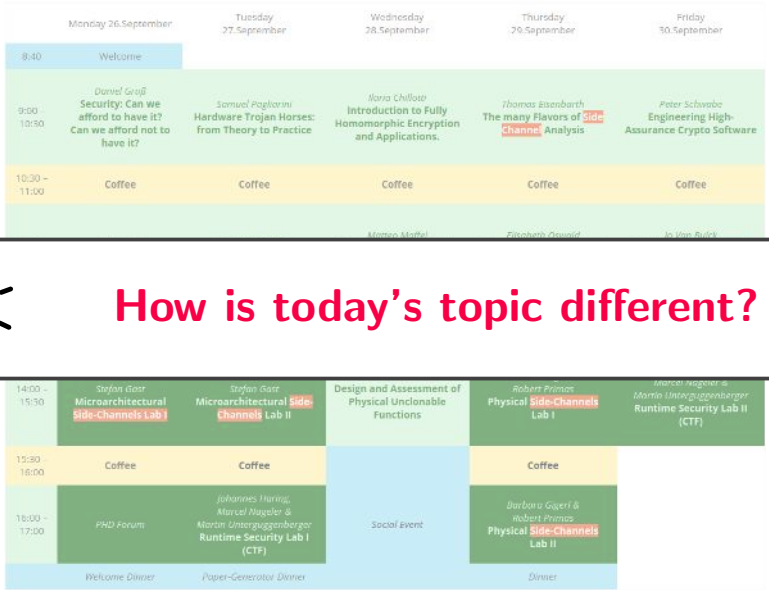
KU LEUVEN

Yet another side-channel lecture?



	Monday 26.September	Tuesday 27.September	Wednesday 28.September	Thursday 29.September	Friday 30.September
8:40	Welcome				
9:00 – 10:30	*Daniel Gruß* **Security: Can we afford to have it? Can we afford not to have it?**	*Samuel Pagliarini* **Hardware Trojan Horses: from Theory to Practice**	*Ilaria Chillotti* **Introduction to Fully Homomorphic Encryption and Applications.**	*Thomas Eisenbarth* **The many Flavors of Side-Channel Analysis**	*Peter Schwabe* **Engineering High-Assurance Crypto Software**
10:30 – 11:00	**Coffee**	**Coffee**	**Coffee**	**Coffee**	**Coffee**
11:00 – 12:30	*Anders Fogh* **Product Security**	*Andrea Fioraldi* **Modern Fuzzing Research and Engineering**	*Matteo Maffei* **Computer-Aided Formal Security Analysis of the Web Platform**	*Elisabeth Oswald* **Techniques and Schemes to Evaluate Side Channel Resilience**	*Jo Van Bulck* **Privileged Side-Channel Attacks on Trusted Execution Environments**
12:30 – 14:00	**Lunch**	**Lunch**	**Lunch**	**Lunch**	**Lunch**
14:00 – 15:30	*Martin Schwarzl & Stefan Gast* **Microarchitectural Side-Channels Lab I**	*Martin Schwarzl & Stefan Gast* **Microarchitectural Side-Channels Lab II**	*Michael Pehl* **Design and Assessment of Physical Unclonable Functions**	*Barbara Gigerl & Robert Primas* **Physical Side-Channels Lab I**	*Johannes Haring, Marcel Nageler & Martin Unterguggenberger* **Runtime Security Lab II (CTF)**
15:30 – 16:00	**Coffee**	**Coffee**		**Coffee**	
16:00 – 17:00	*PhD Forum*	*Johannes Haring, Marcel Nageler & Martin Unterguggenberger* **Runtime Security Lab I (CTF)**	*Social Event*	*Barbara Gigerl & Robert Primas* **Physical Side-Channels Lab II**	
	Welcome Dinner	*Paper-Generator Dinner*		*Dinner*	

Yet another side-channel lecture?



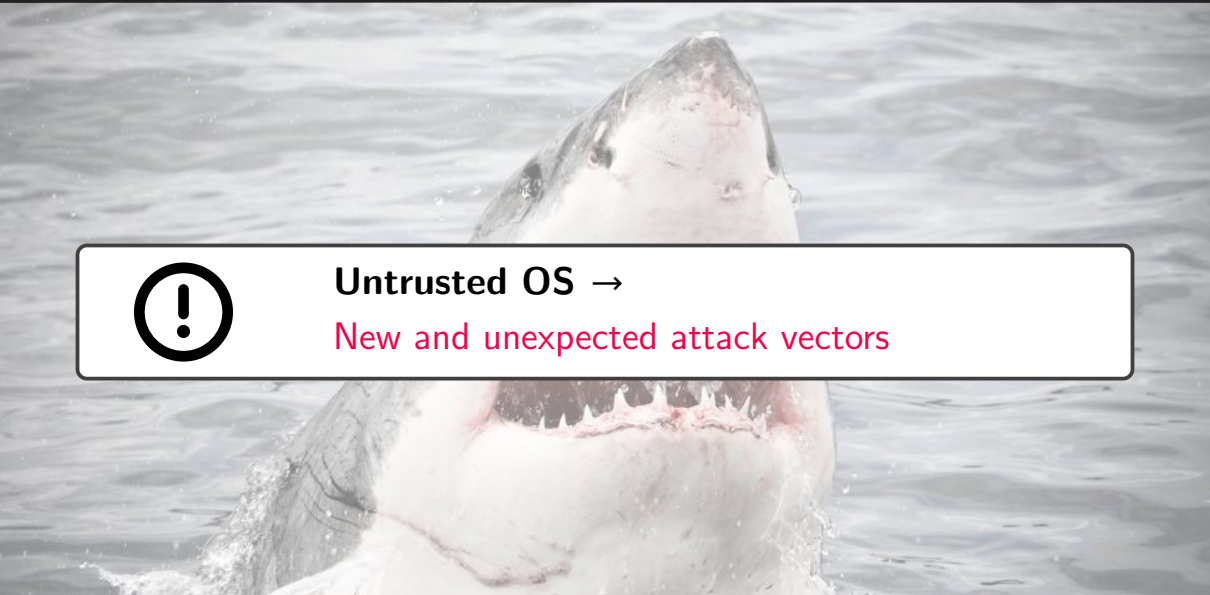
How is today's topic different?

Traditional “top-down” adversary model . . .

Photograph by Hal Schmitt

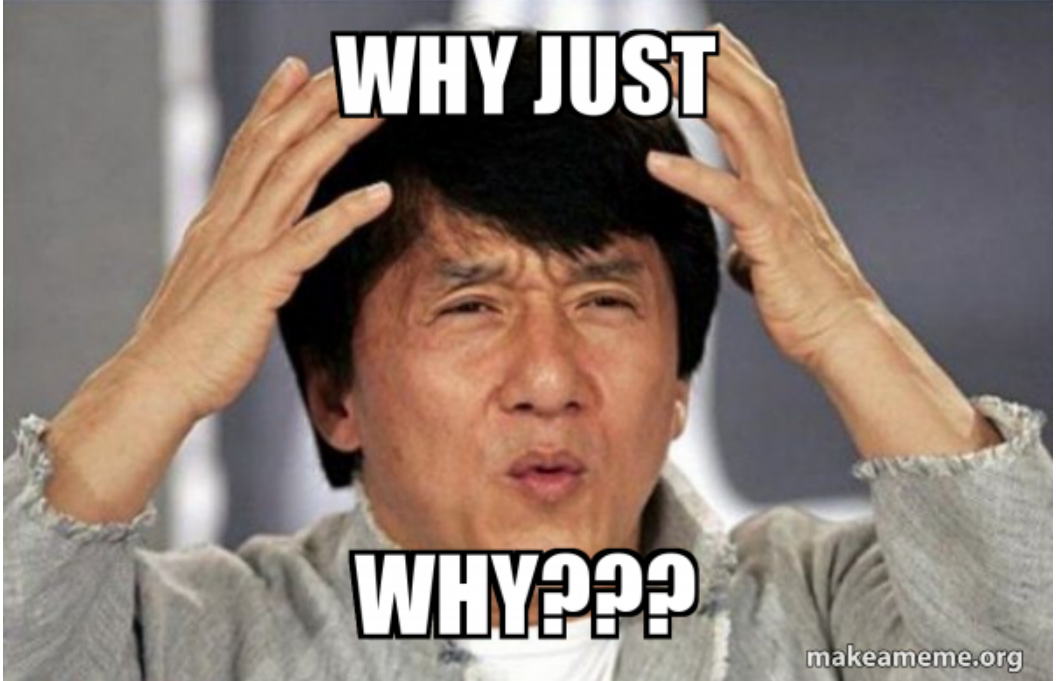
Privileged “bottom-up” adversary model

Privileged “bottom-up” adversary model



Untrusted OS →

New and unexpected attack vectors


WHY JUST
WHY???
makeameme.org

The rise of trusted execution environments

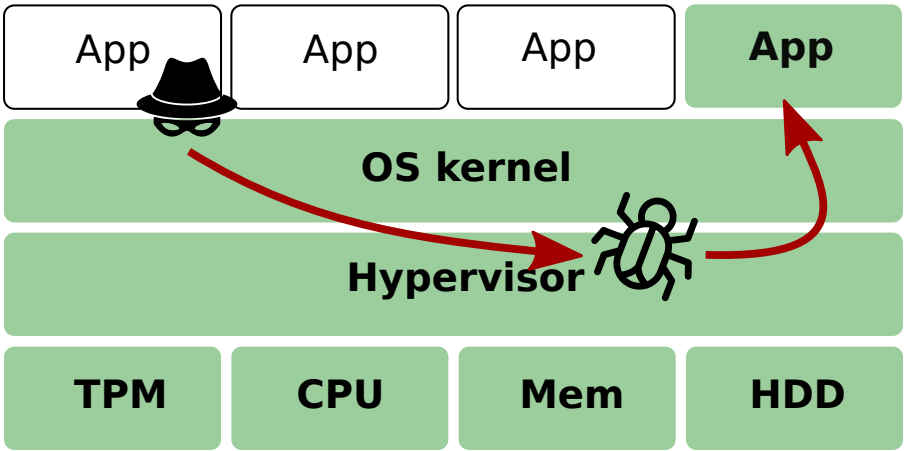
- 2004: ARM TrustZone
- 2015: Intel Software Guard Extensions (SGX)
- 2016: AMD Secure Encrypted Virtualization (SEV)
- 2017: AMD SEV with Encrypted State (SEV-ES)
- 2018: IBM Protected Execution Facility (PEF)
- 2020: AMD SEV with Secure Nested Paging (SEV-SNP)
- 2022: Intel Trust Domain Extensions (TDX)

The rise of trusted execution environments

- 2004: ARM TrustZone
- 2015: Intel Software Guard Extensions (SGX)
- 2016: AMD Secure Encrypted Virtualization (SEV)
- 2017: AMD SEV with Encrypted State (SEV-ES)
- 2018: IBM Protected Execution Facility (PEF)
- 2020: AMD SEV with Secure Nested Paging (SEV-SNP)
- 2022: Intel Trust Domain Extensions (TDX)

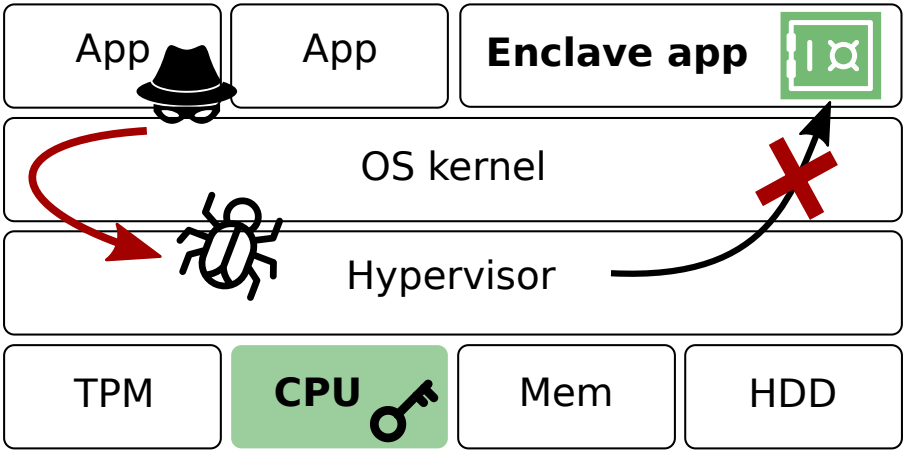
TEEs are here to stay...

The big picture: Reducing attack surface with enclaves



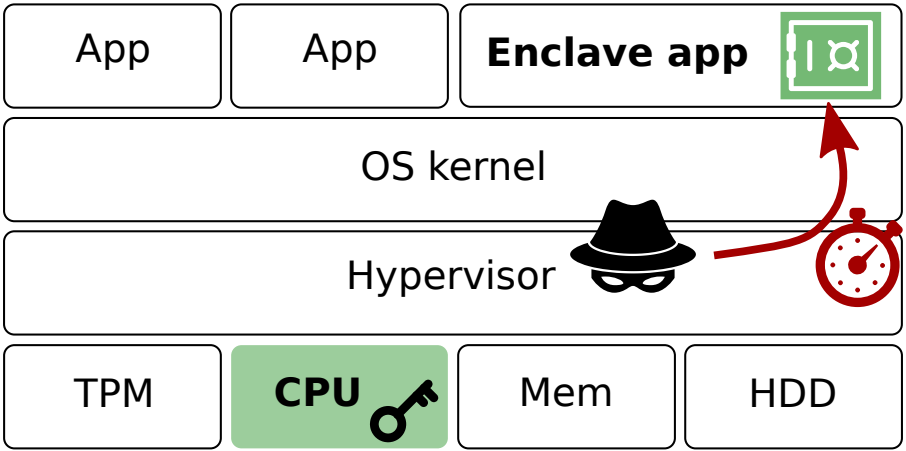
Traditional **layered designs:** Large trusted computing base

The big picture: Reducing attack surface with enclaves



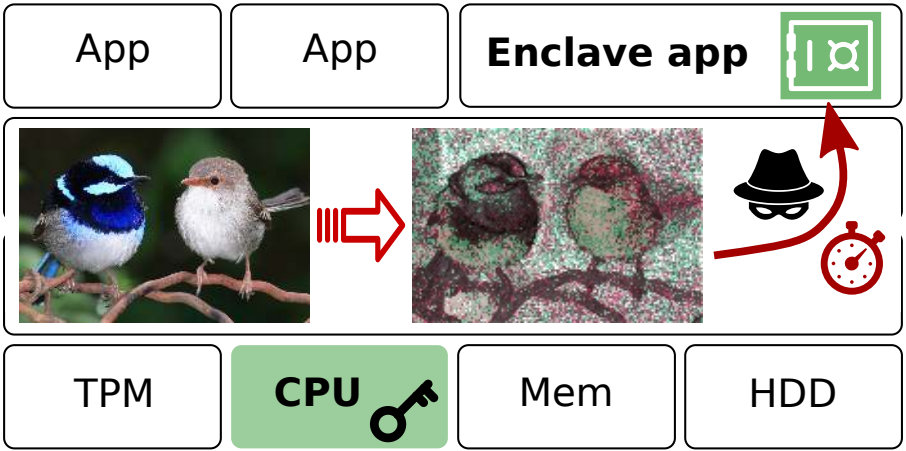
Intel SGX promise: Hardware-level **isolation and attestation**

The big picture: Privileged side-channel attacks



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

The big picture: Privileged side-channel attacks



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


NOT REALLY RELEVANT
THOUGH IS IT?
memegenerator.net

How Trusted Execution Environments Fuel Research on Microarchitectural Attacks

Michael Schwarz, Daniel Gruss
Graz University of Technology, Austria
michael.schwarz@iaik.tugraz.at, daniel.gruss@iaik.tugraz.at

***Abstract*—Trusted Execution Environments (TEEs) enabled research in scenarios with highest-privileged attackers with precise control over system and microarchitecture. Insights gained from such attacks facilitated the discovery of non-TEE attacks like Spectre and Foreshadow from within virtual machines. Future research on microarchitectural attacks will continue to draw motivation from TEE threat models.**

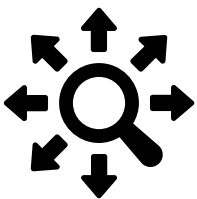
https://gruss.cc/files/tee_fuel.pdf

TEE attack research leads the way . . .

TEE attack research leads the way . . .

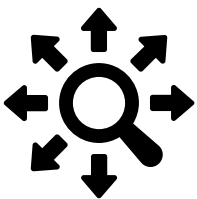
- Privileged TEE attacker models sets the bar!
- Idealized execution environment for attack research
- **Generalizations:** e.g., Foreshadow-NG, branch prediction, address translation, etc.

Research agenda: Understanding privileged attack surface



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

Research agenda: Understanding privileged attack surface

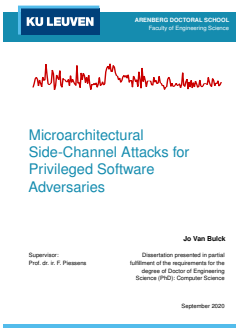


1. **Which** novel privileged attacks exist?
   → Uncover previously unknown attack avenues

2. **How** well can they be exploited in practice?
   → Develop new techniques and practical attack frameworks

3. **What** can be leaked?
   → Leak metadata and data

Research agenda: Understanding privileged attack surface


KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

Microarchitectural Side-Channel Attacks for Privileged Software Adversaries

Jo Van Bulck

Supervisor:
Prof. dr. ir. F. Piessens

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Computer Science

September 2020

1. **Which** novel privileged attacks exist?
   → Uncover previously unknown attack avenues

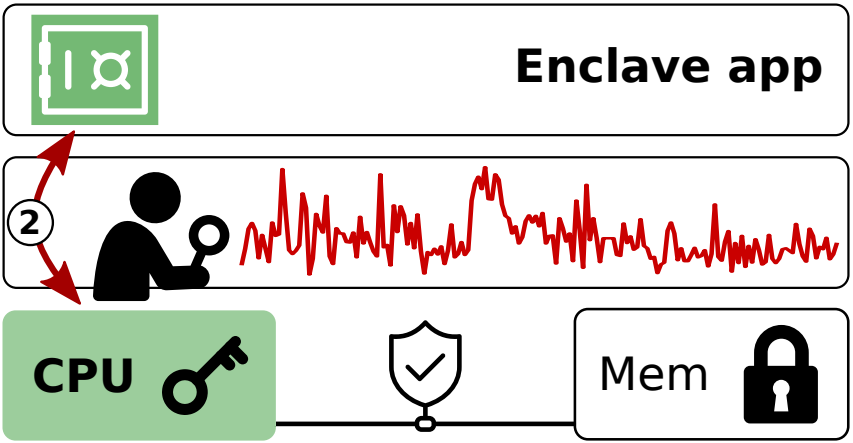
2. **How** well can they be exploited in practice?
   → Develop new techniques and practical attack frameworks

3. **What** can be leaked?
   → Leak metadata and data

Jo Van Bulck, "Microarchitectural Side-Channel Attacks for Privileged Software Adversaries", PhD thesis KU Leuven, 2020.

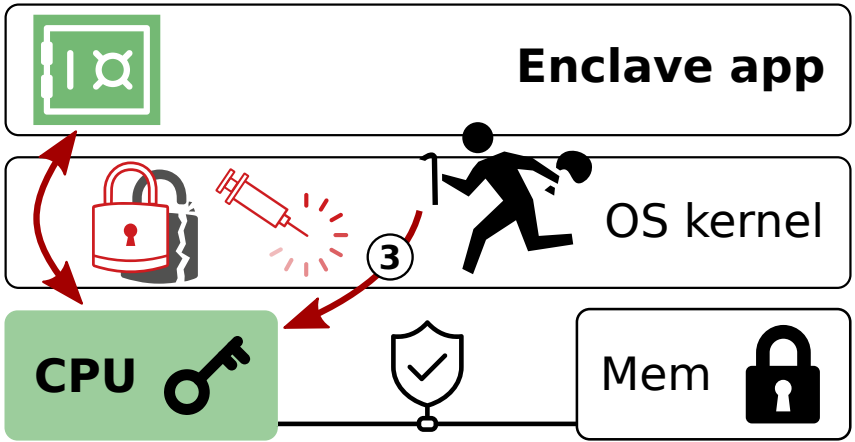
https://jovanbulck.github.io/files/phd-thesis.pdf

Lecture overview: Privileged side-channel attacks (part 1)



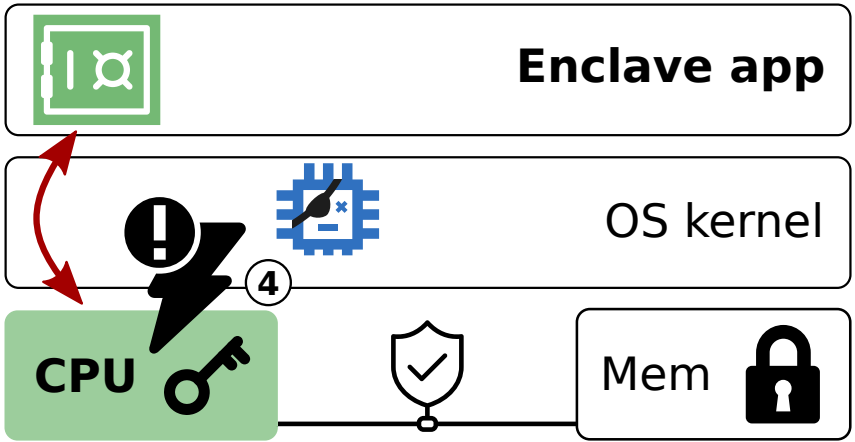
Privileged side channels to spy on enclave-CPU interaction metadata

Lecture overview: Transient-execution attacks (part 2)



Transient-execution data extraction from CPU to break enclave confidentiality

Lecture overview: Privileged CPU interface attacks (part 3)



Unexpected (physical) **CPU interface** manipulations to break enclave integrity

Attack vector 1: Side-channel analysis

A note on SGX side-channel attacks (Intel)

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

KEEP CALM

IT IS

OUT OF SCOPE

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 }
```

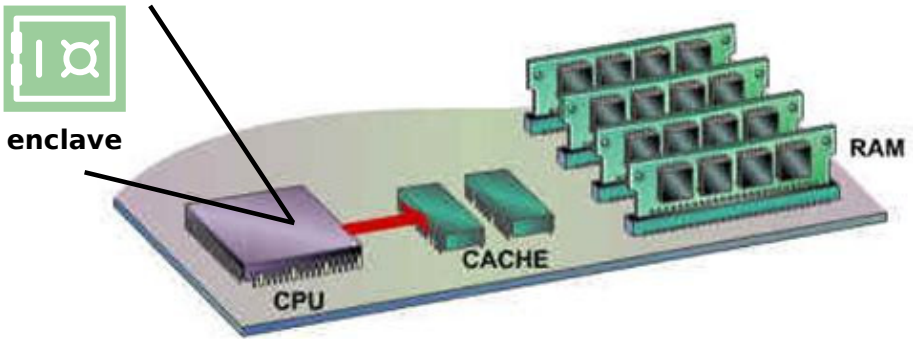
Vulnerable patterns: Secret-dependent code/data accesses

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

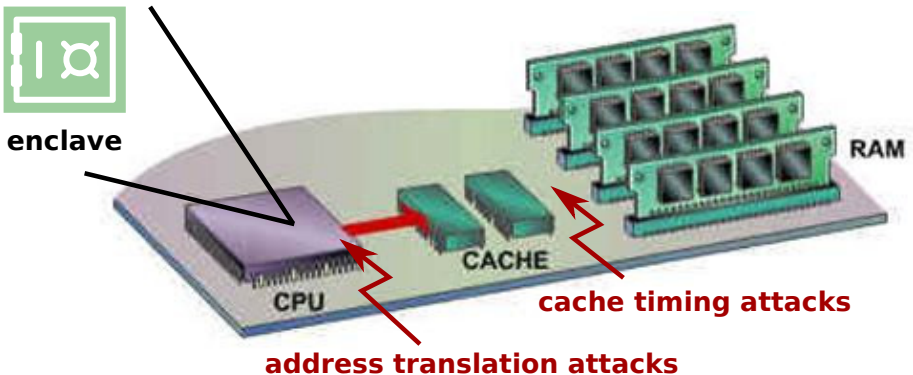
```
1 int secret_lookup(int s)
2 {
3     if (s > 0 && s < ARRAY_LEN)
4         return array[s];
5     return -1;
6
7 }
```

What are new ways for privileged adversaries to create an "oracle" for enclave code+data memory accesses?

Overview: Spying on enclave memory accesses



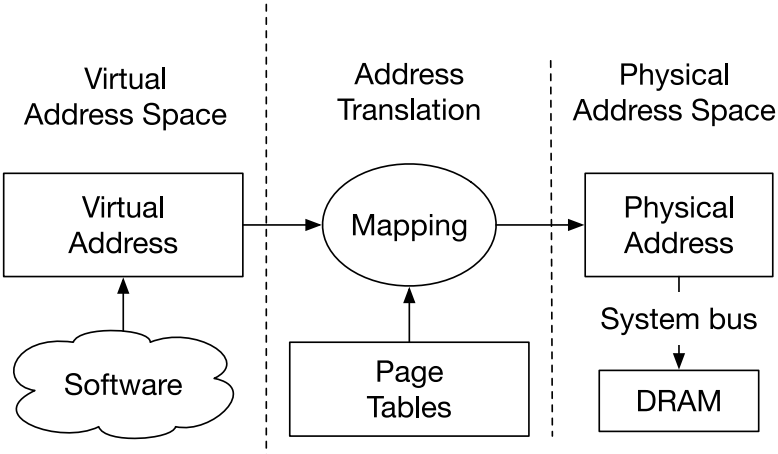
Overview: Spying on enclave memory accesses



Attack vector 1: Side-channel analysis

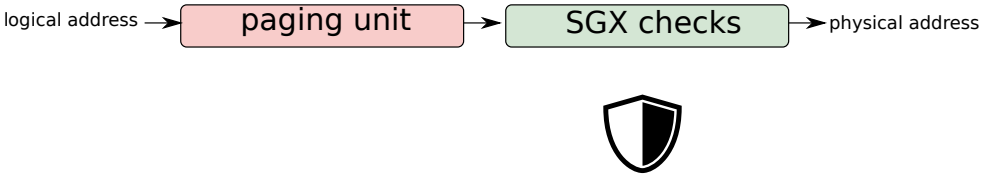
Idea #1: Monitoring address translation?

The virtual memory abstraction



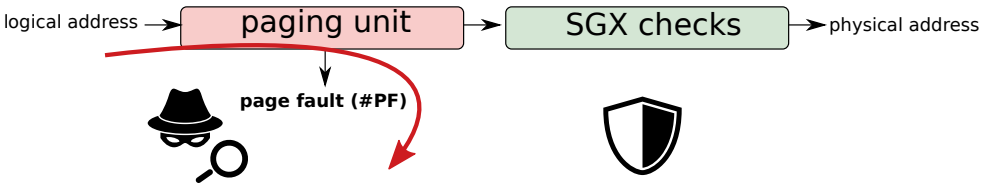
Costan et al. "Intel SGX explained", IACR 2016.

Intel SGX: Page faults as a side channel



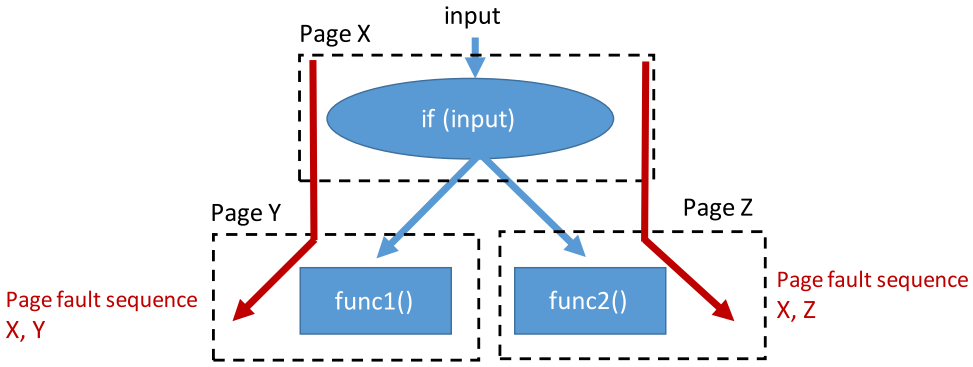
SGX machinery protects against direct address remapping attacks

Intel SGX: Page faults as a side channel



. . . but untrusted address translation may **fault(!)**

Intel SGX: Page faults as a side channel

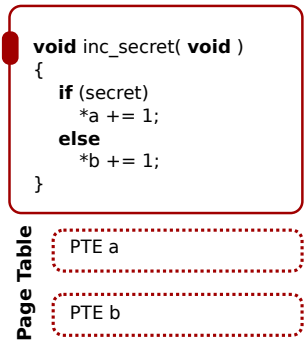


Xu et al.: "Controlled-channel attacks: Deterministic side channels for untrusted operating systems", Oakland 2015.

⇒ Page fault traces leak **private control data/flow**

An end-to-end example

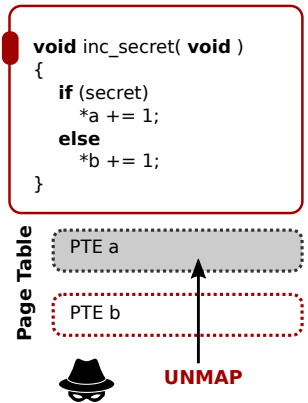
https://github.com/jovanbulck/sgx-tutorial-space18



An end-to-end example

https://github.com/jovanbulck/sgx-tutorial-space18

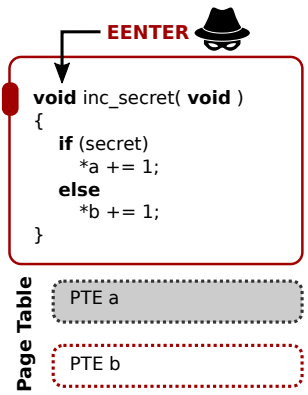
1. Revoke access rights on *unprotected* enclave page table entry



An end-to-end example

https://github.com/jovanbulck/sgx-tutorial-space18

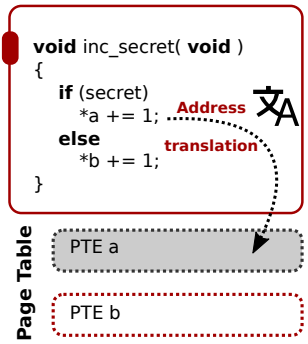
1. Revoke access rights on *unprotected* enclave page table entry
2. Enter victim enclave



An end-to-end example

https://github.com/jovanbulck/sgx-tutorial-space18

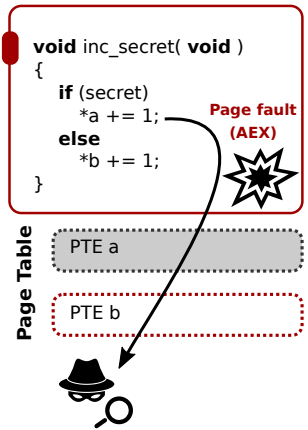
1. Revoke access rights on *unprotected* enclave page table entry
2. Enter victim enclave
3. Secret-dependent data memory access
   - ⇝ CPU performs virt-to-phys address translation!
   - ⇝ By reading page table entry setup by *untrusted* OS



An end-to-end example

https://github.com/jovanbulck/sgx-tutorial-space18

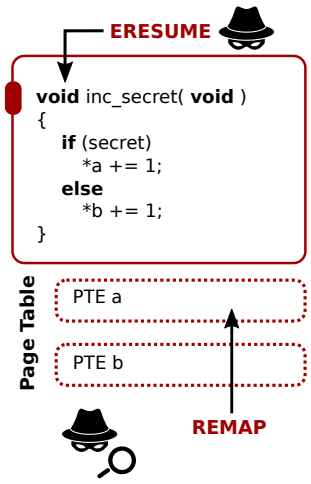
1. Revoke access rights on *unprotected* enclave page table entry
2. Enter victim enclave
3. Secret-dependent data memory access
   - ⇝ CPU performs virt-to-phys address translation!
   - ⇝ By reading page table entry setup by *untrusted* OS
4. Virtual address not present → raise page fault
   - ⇝ CPU exits enclave and vectors to untrusted OS



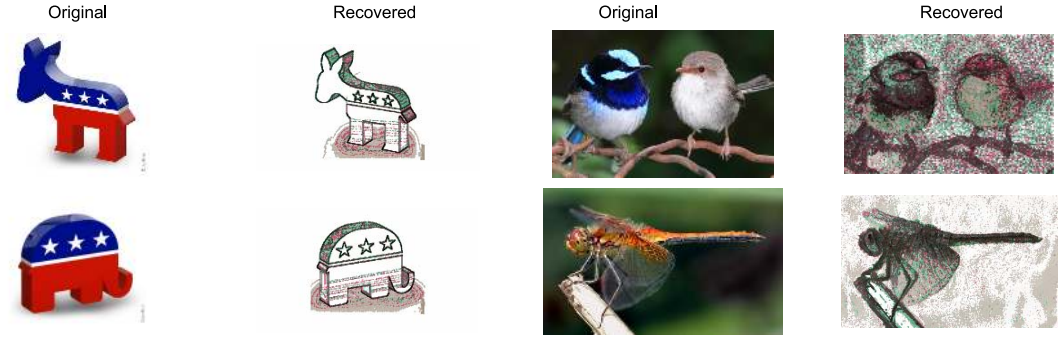
An end-to-end example

https://github.com/jovanbulck/sgx-tutorial-space18

1. Revoke access rights on *unprotected* enclave page table entry
2. Enter victim enclave
3. Secret-dependent data memory access
   - ⇝ CPU performs virt-to-phys address translation!
   - ⇝ By reading page table entry setup by *untrusted* OS
4. Virtual address not present → raise page fault
   - ⇝ CPU exits enclave and vectors to untrusted OS
5. Restore access rights and resume victim enclave



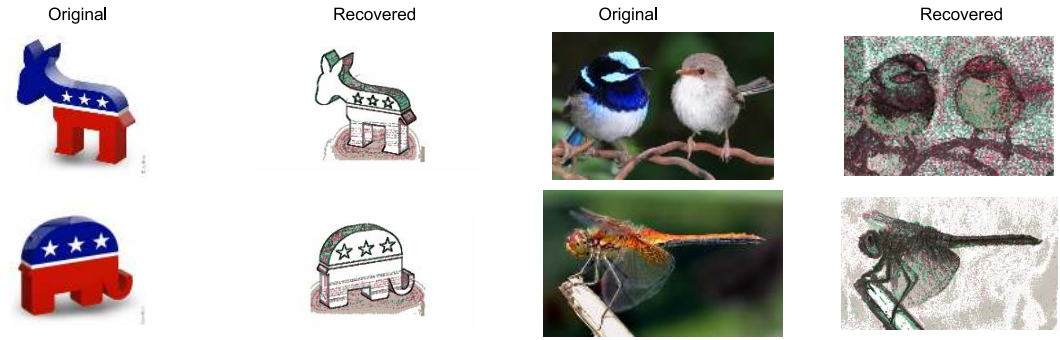
Page table-based attacks in practice



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



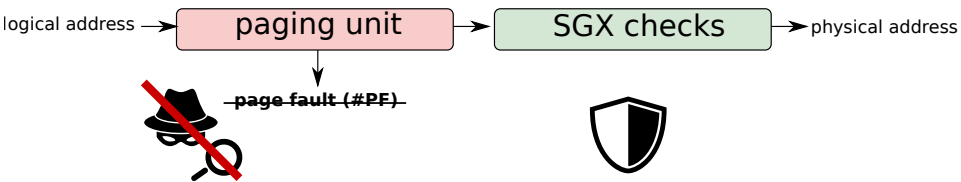
Xu et al.: "Controlled-channel attacks: Deterministic side channels for untrusted operating systems", Oakland 2015.

. . . but **many faults** and a coarse-grained **4 KiB granularity**

Attack vector 1: Side-channel analysis

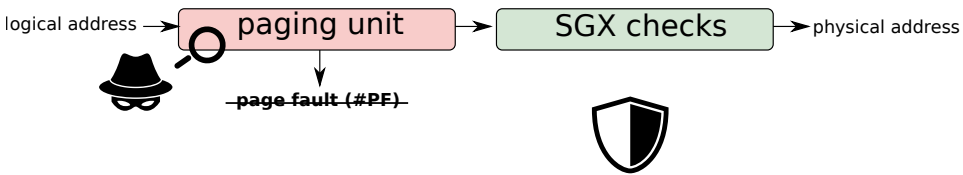
Idea #2: Monitoring without page faults?

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 learn page visits without triggering faults!

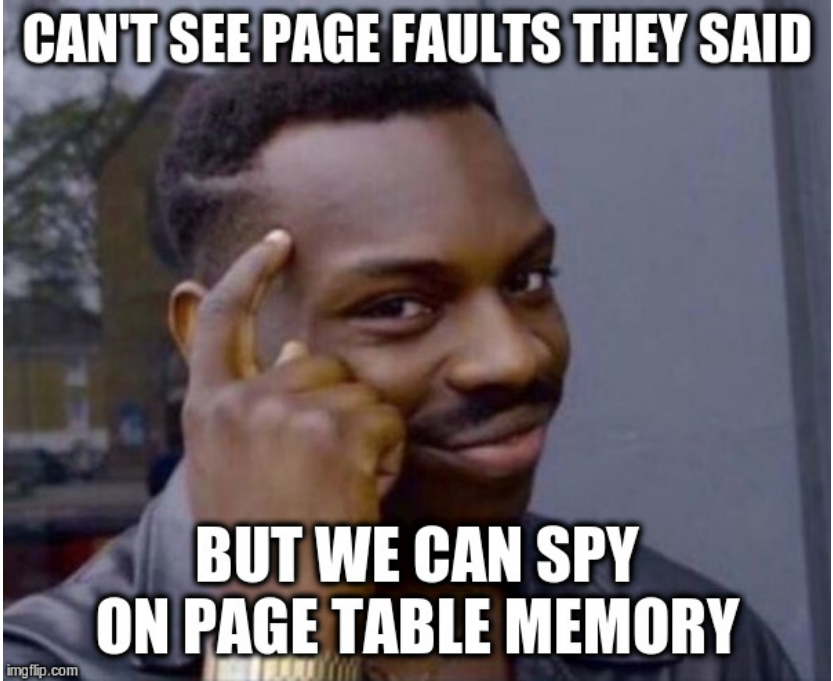
Documented side-effects of address translation

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).

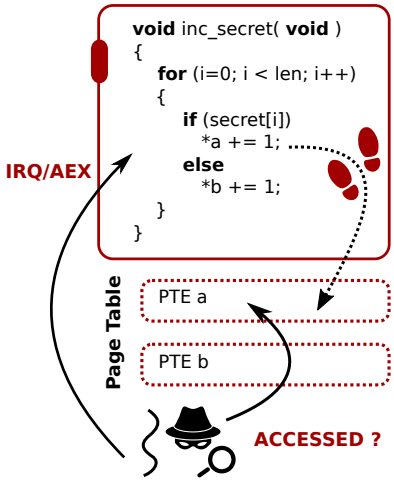
Whenever there is a write to a linear address, the processor sets the dirty flag (if it is not already set) in the paging-structure entry that identifies the final physical address for the linear address (either a PTE or a paging-structure entry in which the PS flag is 1).


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.

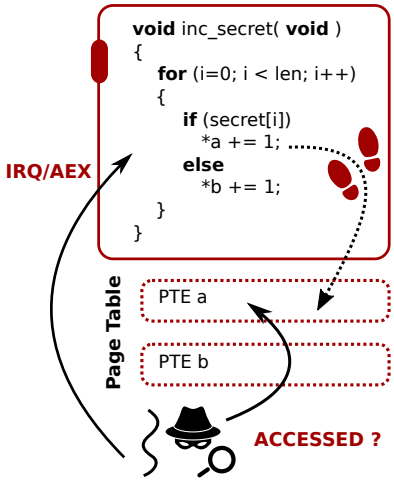
Telling your secrets without page faults

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!

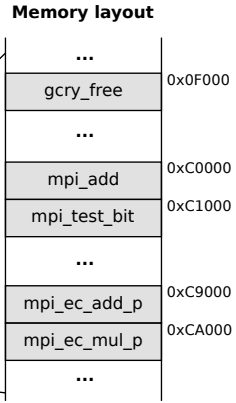


Van Bulck et al. "Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution", USENIX 2017.

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 }
```

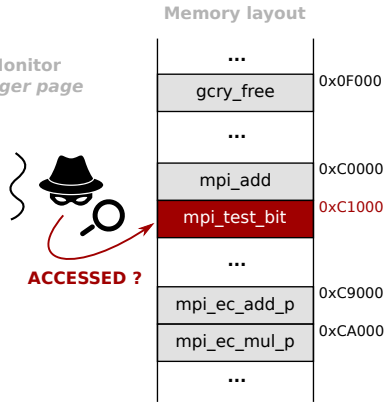
22 Code pages per iteration



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 }
```



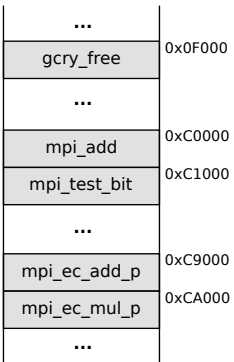
Van Bulck et al. "Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution", USENIX 2017.

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 }
```

INTERRUPT

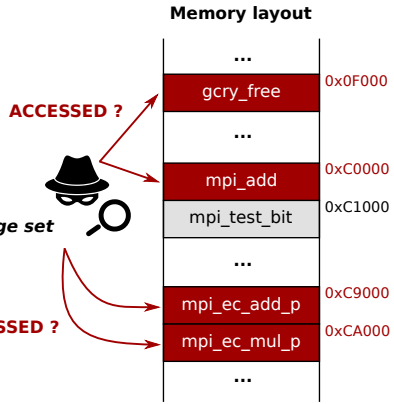
Memory layout



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.

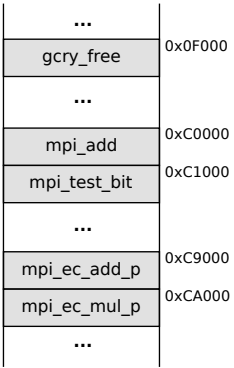
Attacking Libgcrypt EdDSA (simplified)

```
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 }
```

RESUME

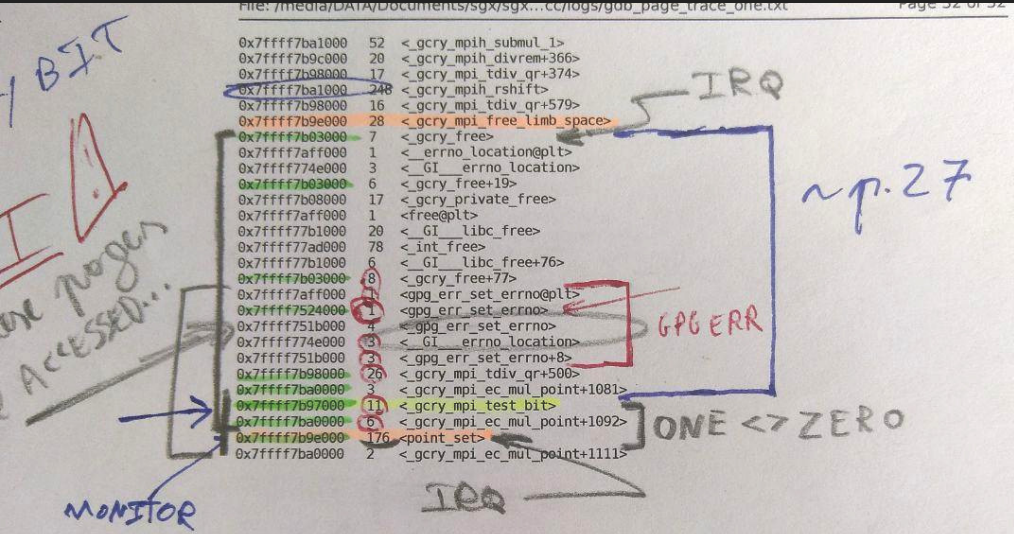
Full 512-bit key recovery, single run

Memory layout



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

Side-channel analysis: From metadata patterns to secrets



Van Bulck et al. "Telling Your Secrets Without Page Faults: Stealthy Page Table-Based Attacks on Enclaved Execution", USENIX 2017.

Scientific understanding driven by attacker-defender race . . .

Attack vector 1: Side-channel analysis

Idea #3: Spatial vs. temporal resolution?

Intel's note on side-channel attacks (revisited)

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

Temporal resolution limitations for the page fault oracle

```c
size_t strlen (char *str)
{
  char *s;

  for (s = str; *s; ++s);
  return (s - str);
}
```

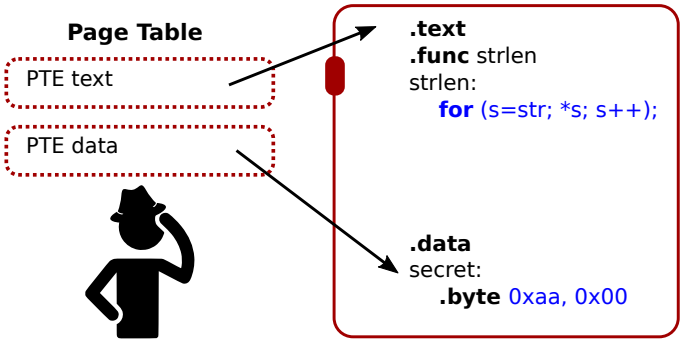
```
    mov  %rdi,%rax
1:  cmpb $0x0,(%rax)
    je   2f
    inc  %rax
    jmp  1b
2:  sub  %rdi,%rax
    retq
```

⇒ tight loop: 4 instructions, single memory operand, single code + data page

Counting `strlen` loop iterations?

Note: Page-fault attacks cannot make progress for 1 code + data page

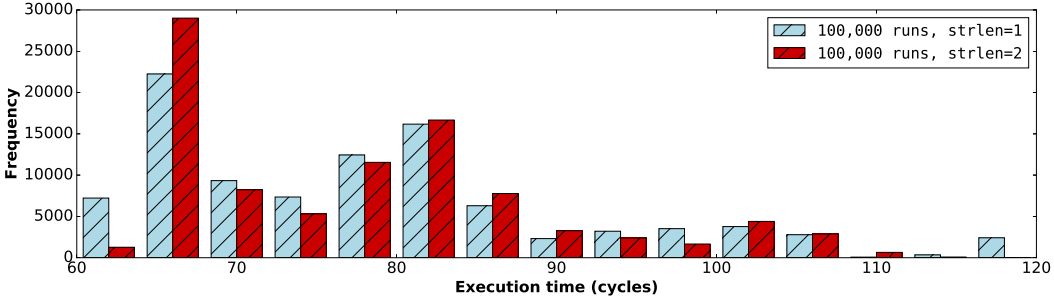
Temporal resolution limitations for the page fault oracle



Counting `strlen` loop iterations?

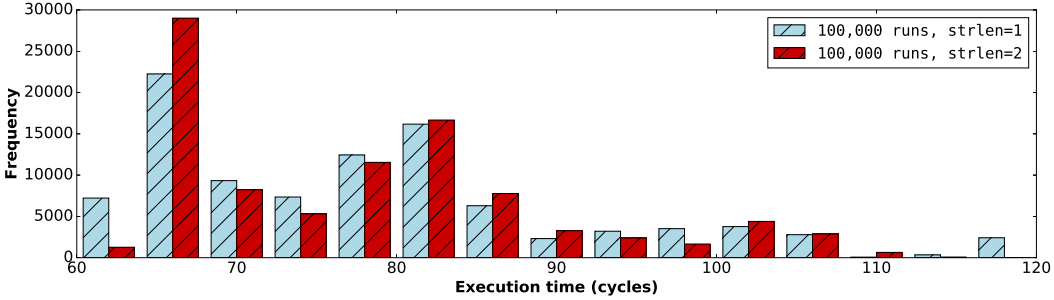
Progress requires both pages present (non-faulting) ↔ page fault oracle

Building the `strlen()` side-channel oracle with execution timing?

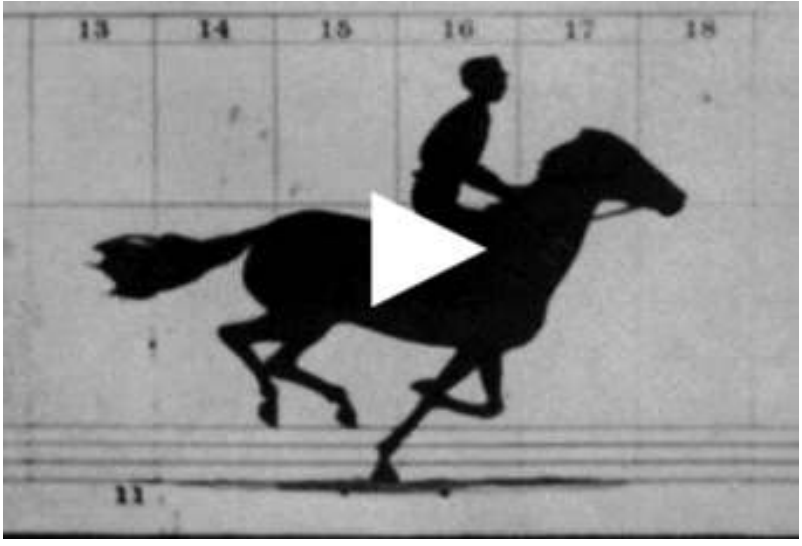


Building the `strlen()` side-channel oracle with execution timing?

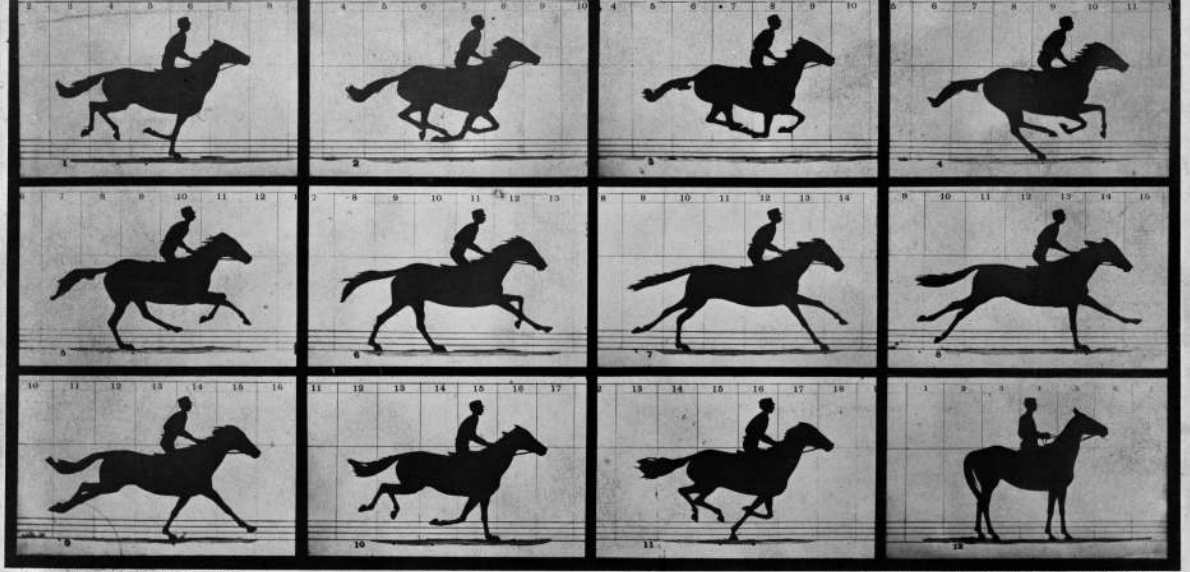
Too noisy: modern x86 processors are lightning fast...



Analogy: Studying galloping horse dynamics



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



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

THE HORSE IN MOTION.

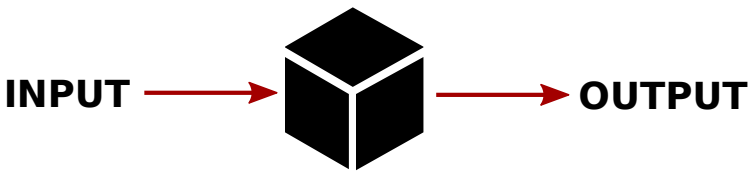
Illustrated by

MUYBRIDGE.

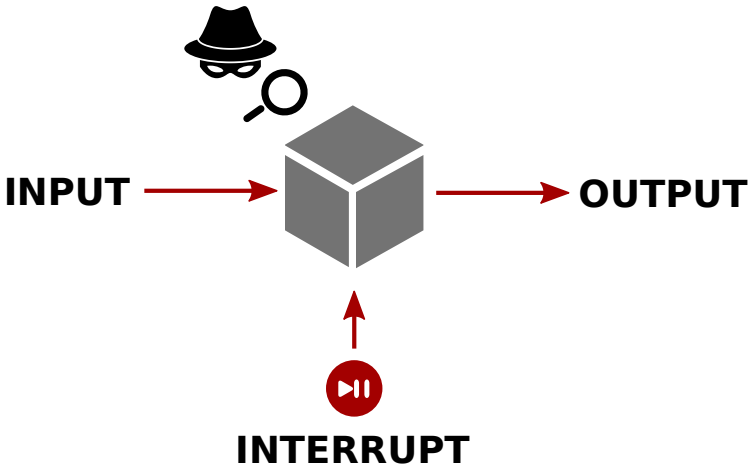
AUTOMATIC ELECTRO-PHOTOGRAPH.

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

SGX-Step: Executing enclaves one instruction at a time

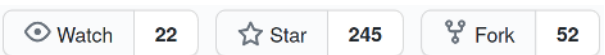


SGX-Step: Executing enclaves one instruction at a time



SGX-Step: Executing enclaves one instruction at a time

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



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:

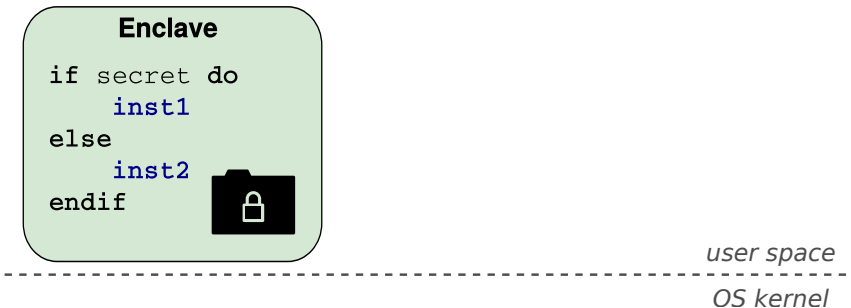
☹ x86 hardware *debug features* disabled in enclave mode

☺ . . . but we have *root access!*

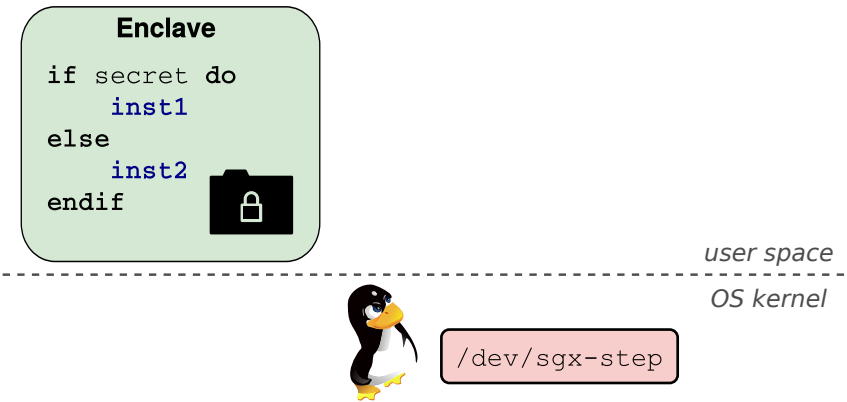
⇒ Setup user-space virtual **memory mappings** for x86 APIC (+ PTEs)

```
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:~$
```

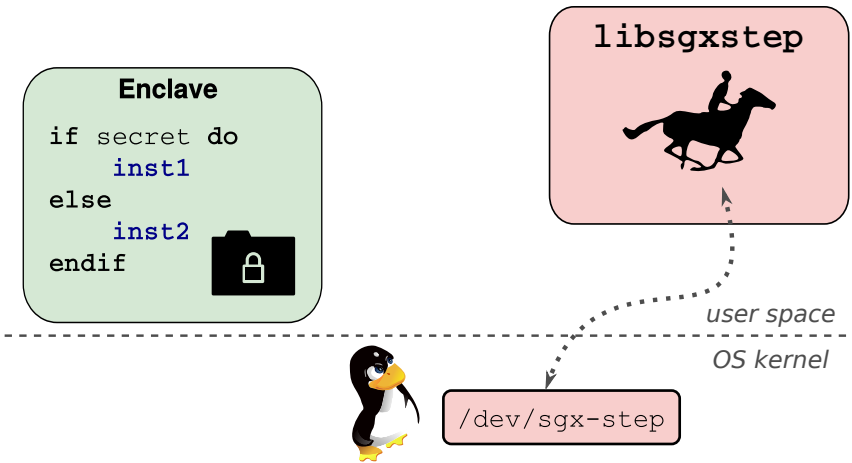
SGX-Step: Executing enclaves one instruction at a time



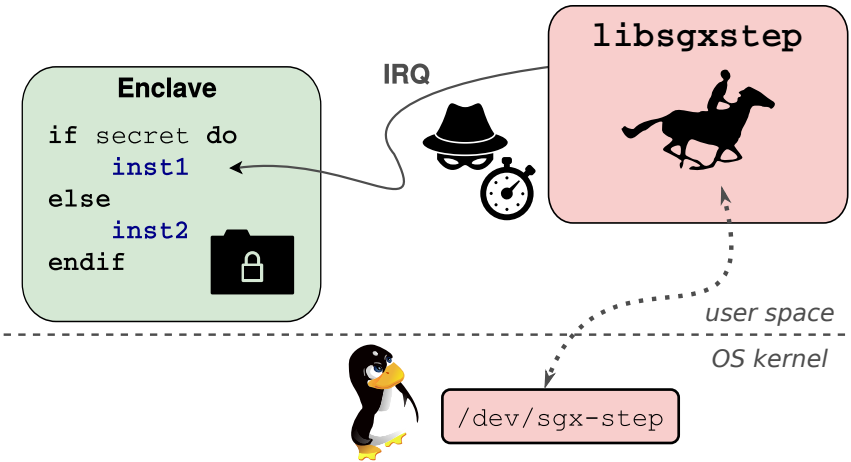
SGX-Step: Executing enclaves one instruction at a time



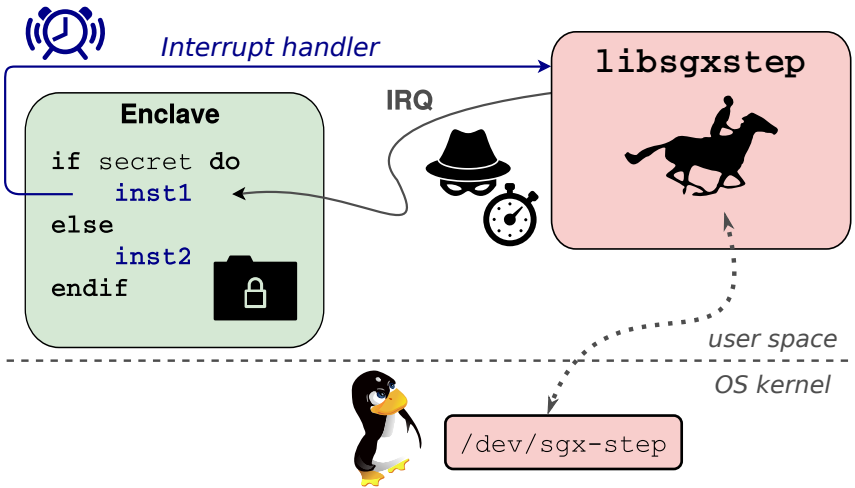
SGX-Step: Executing enclaves one instruction at a time



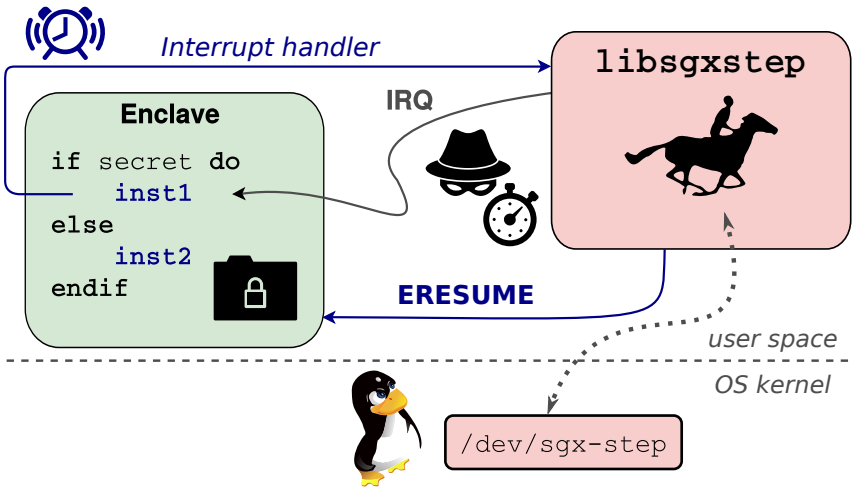
SGX-Step: Executing enclaves one instruction at a time



SGX-Step: Executing enclaves one instruction at a time



SGX-Step: Executing enclaves one instruction at a time

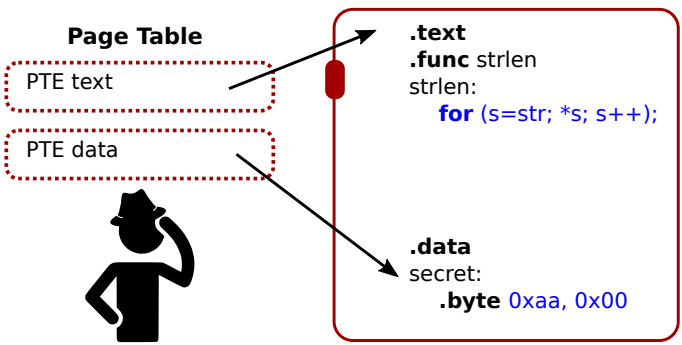


SGX-Step demo: Building a `memcmp()` password oracle

```
[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$
```

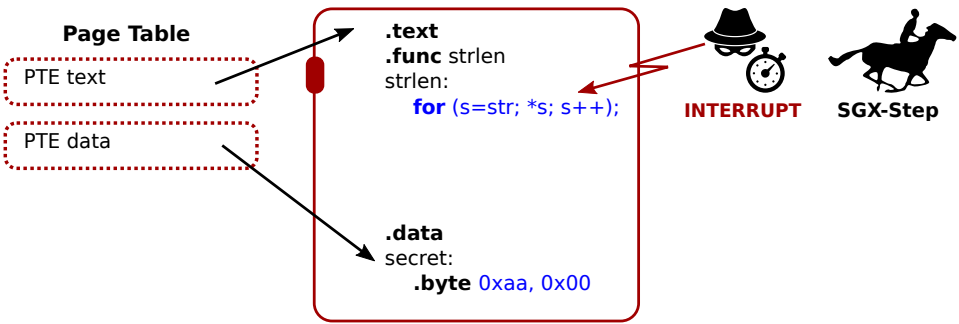
Counting `strlen()` loop iterations with SGX-Step

Progress requires both pages present...



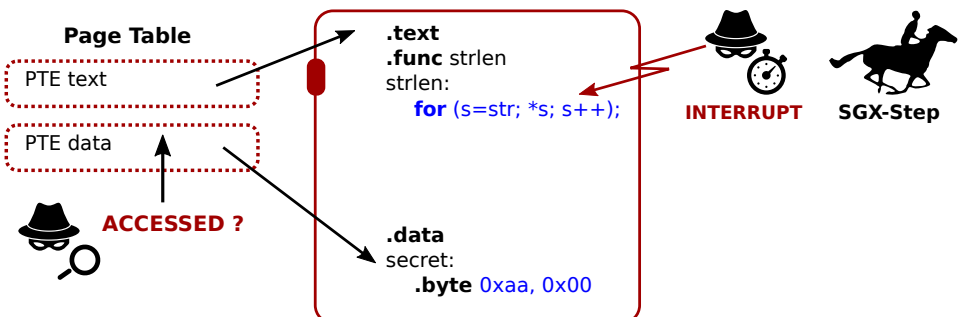
Counting `strlen()` loop iterations with SGX-Step

Execute *exactly* one enclave instruction → **timer interrupt**



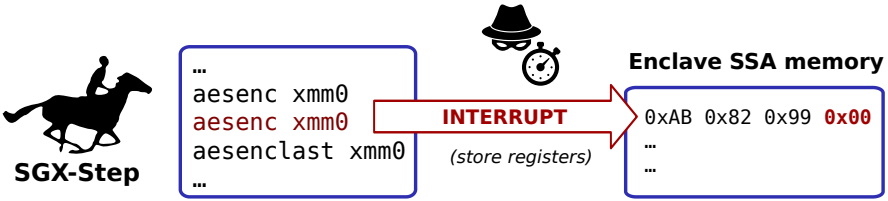
Counting `strlen()` loop iterations with SGX-Step

Page table accessed bit set? → **strlen++** → resume



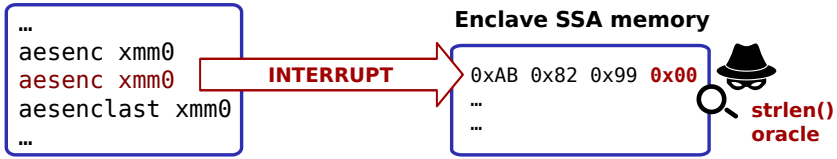

CVE-2018-3626: ALL YOUR ZERO BYTES
ARE BELONG TO US
imgflip.com

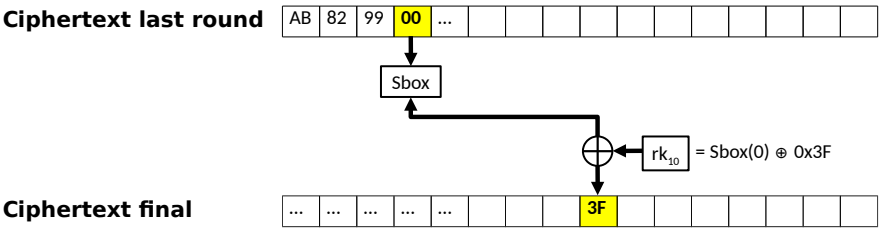
SGX-Step: Breaking AES-NI with the `strlen()` null byte oracle



Van Bulck et al. "A Tale of Two Worlds: Assessing the Vulnerability of Enclave Shielding Runtimes", CCS 2019..

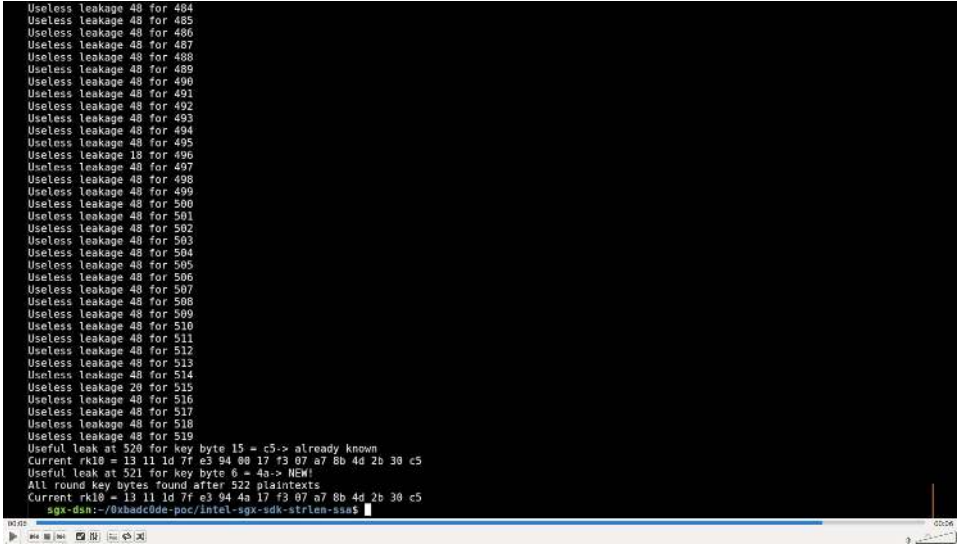
SGX-Step: Breaking AES-NI with the `strlen()` null byte oracle





Van Bulck et al. "A Tale of Two Worlds: Assessing the Vulnerability of Enclave Shielding Runtimes", CCS 2019..

SGX-Step: Breaking AES-NI with the `strlen()` null byte oracle



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

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

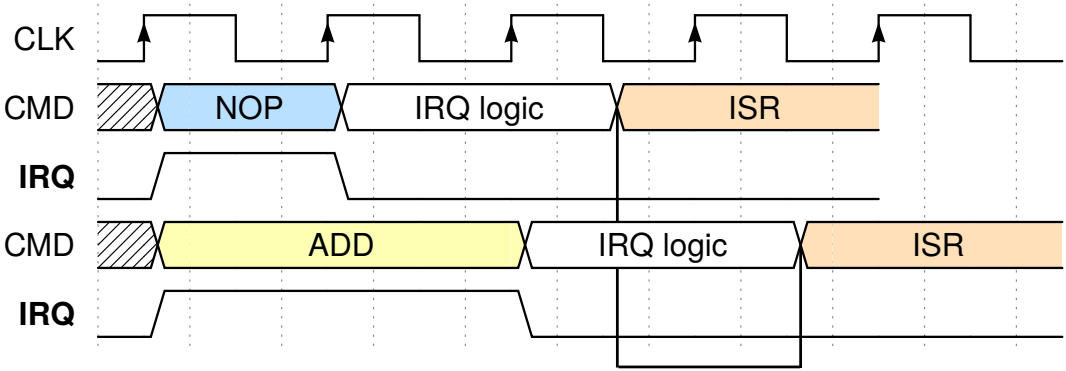
Yr	Attack	Temporal resolution	APIC		PTE			Desc		Drv
			IRQ	IPI	#PF	A/D	PPN	GDT	IDT	
'19	Spoiler	✓ 1	●	○	○	●	○	○	●	✓
'19	ZombieLoad	✓ 0 - 1	●	○	●	●	○	○	●	✓
'19	Tale of 2 worlds	✓ 1	●	○	●	●	○	○	●	✓
'19	MicroScope	~ 0 - Page	○	○	●	○	○	○	○	✗
'20	Bluethunder	✓ 1	●	○	○	○	○	○	●	✓
'20	Big troubles	~ Page	○	○	●	○	○	○	○	✓
'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	○	○	●	○	○	○	○	✓

https://jovanbulck.github.io/files/phd-thesis.pdf

Attack vector 1: Side-channel analysis

Idea #4: Interrupts as a timing leak?

Wait a cycle: Interrupt latency as a side channel

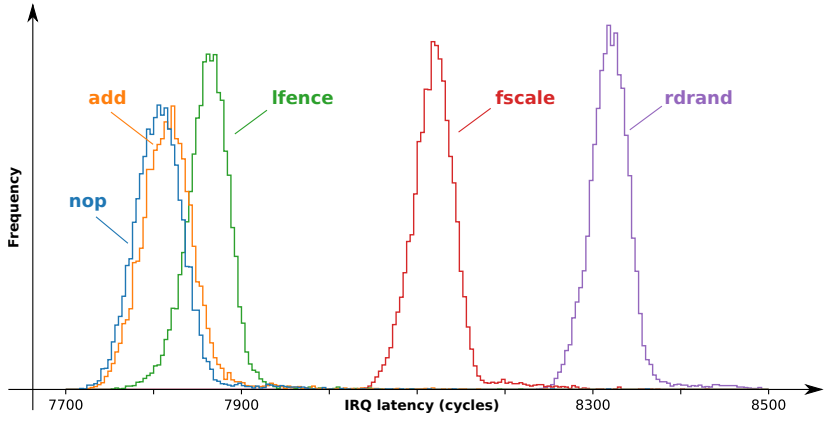



```
if (secret){ ADD @R5+, R6;} // 2 cycles
else       { NOP; NOP;     } // 2*1 cycle
```


TIMING LEAKS
EVERYWHERE
imgflip.com

Nemesis microbenchmarks: Measuring x86 operands

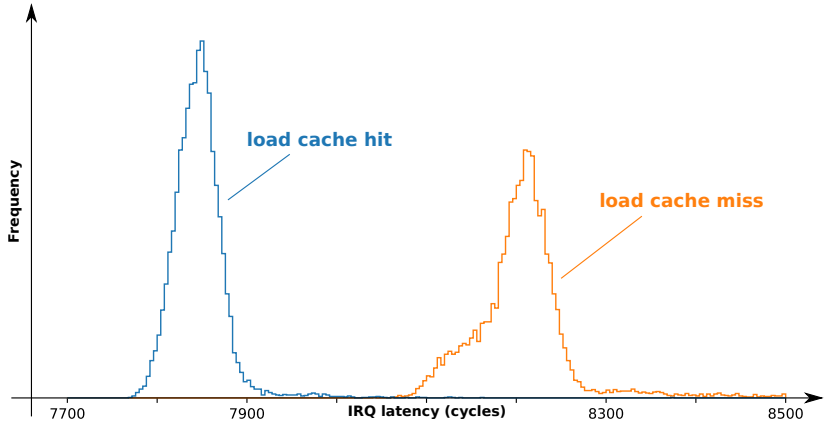
Instruction timing leak: Reconstruct x86 operand class



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

Nemesis microbenchmarks: Measuring x86 cache misses

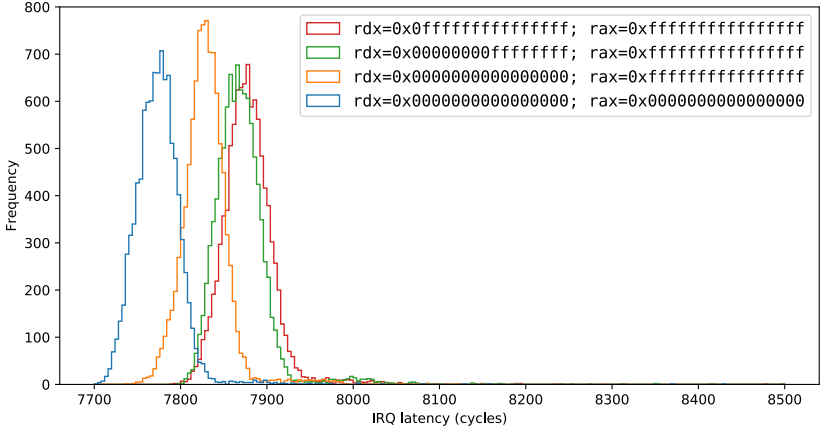
Instruction timing leak: Reconstruct *microarchitectural state*



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

Nemesis microbenchmarks: Measuring x86 data dependencies

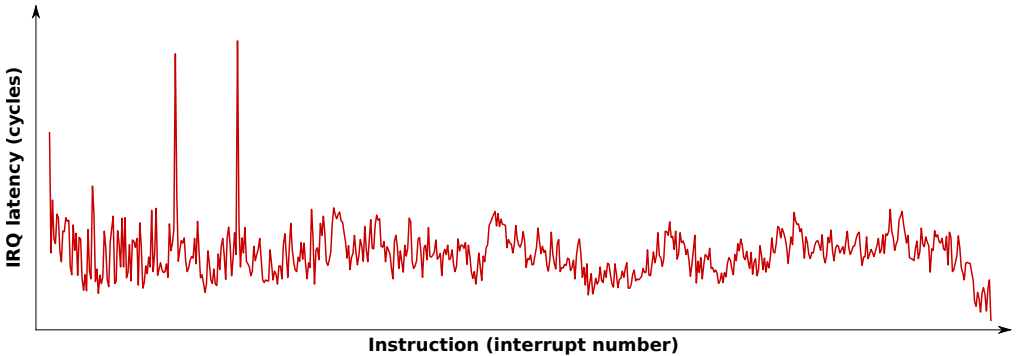
Instruction timing leak: Execution time ≈ dividend significant bits



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

Nemesis: Extracting IRQ latency traces with SGX-Step

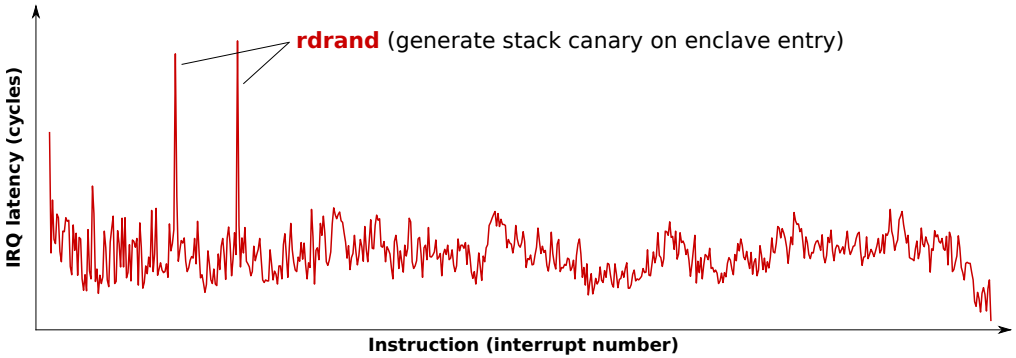
Enclave x-ray: IRQ latency leaks instruction-level μ-arch timing!



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

Nemesis: Extracting IRQ latency traces with SGX-Step

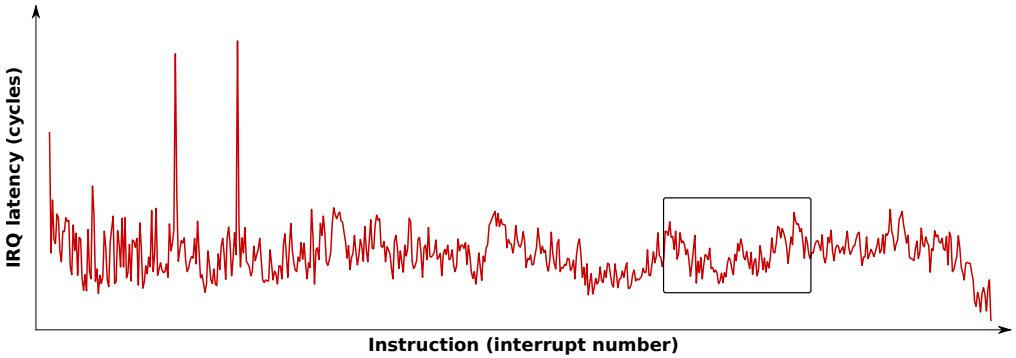
Enclave x-ray: Spotting high-latency instructions



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

Nemesis: Extracting IRQ latency traces with SGX-Step

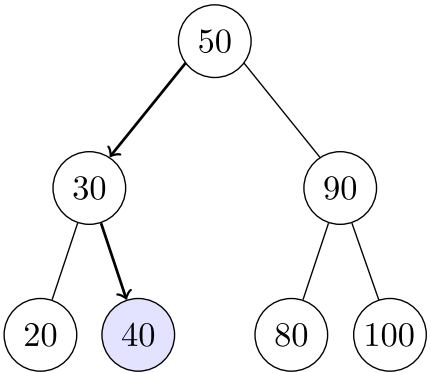
Enclave x-ray: Zooming in on `bsearch` function



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

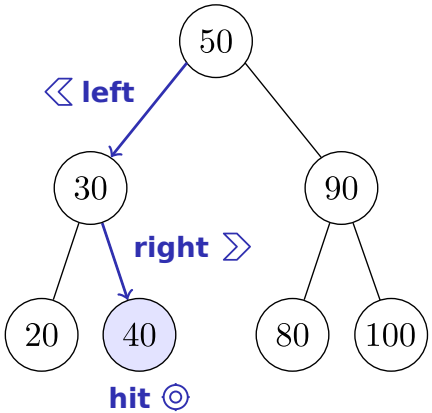
De-anonymizing enclave lookups with interrupt latency

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



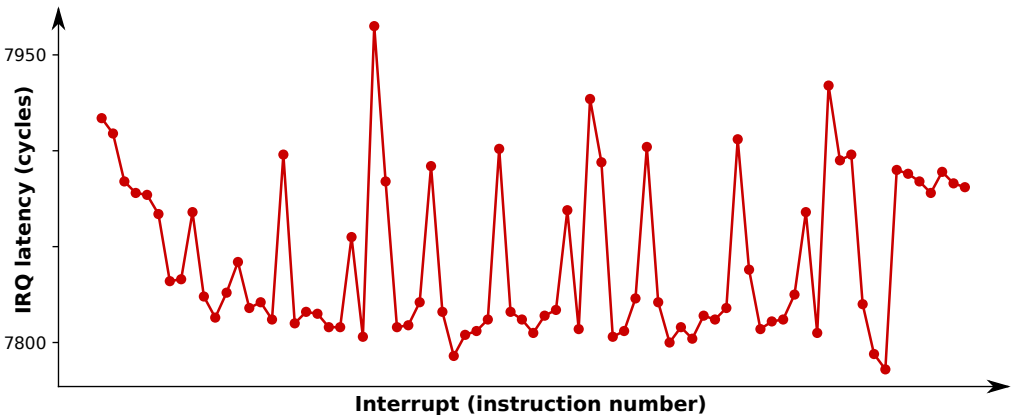
De-anonymizing enclave lookups with interrupt latency

Adversary: Infer secret lookup in known array



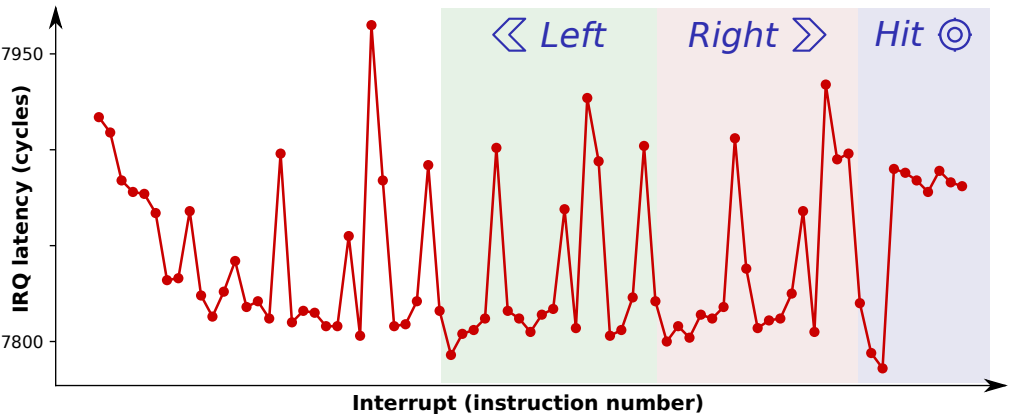
De-anonymizing enclave lookups with interrupt latency

Goal: Infer lookup → reconstruct `bsearch` control flow



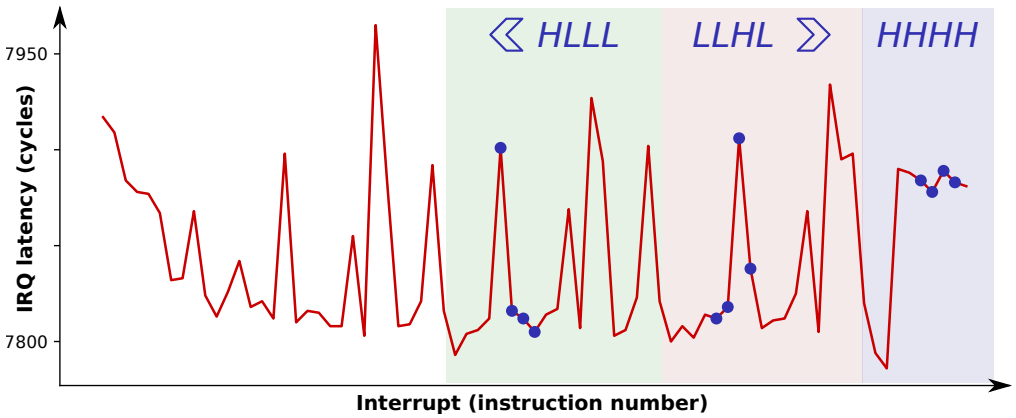
De-anonymizing enclave lookups with interrupt latency

Goal: Infer lookup → reconstruct `bsearch` control flow



De-anonymizing enclave lookups with interrupt latency

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



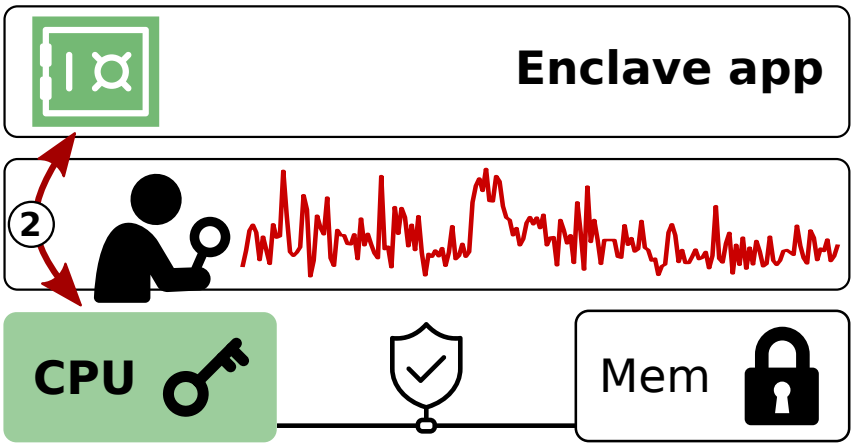
Attack vector 2: Transient execution

Revisited: Intel's note on side-channel attacks (2017)

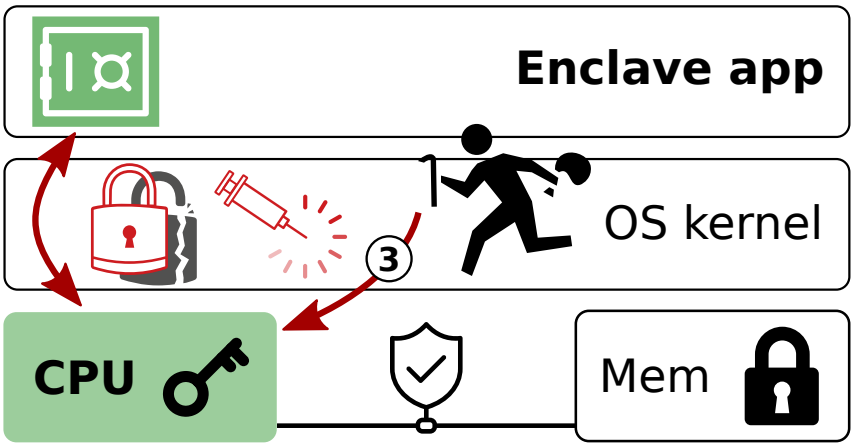
"In general, these research papers do not demonstrate anything new or unexpected about the Intel SGX architecture. Preventing side channel attacks is a matter for the enclave developer. *Intel makes this clear in the security objectives for Intel SGX."*

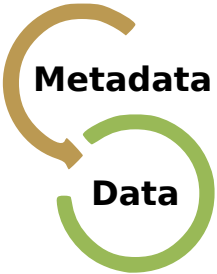
`https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sgx-and-side-channels.html`

Recap: Privileged side-channel attacks



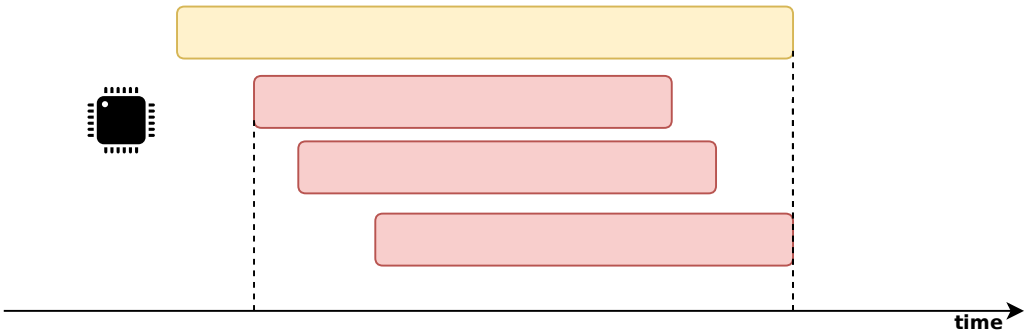
Recap: Transient-execution attacks



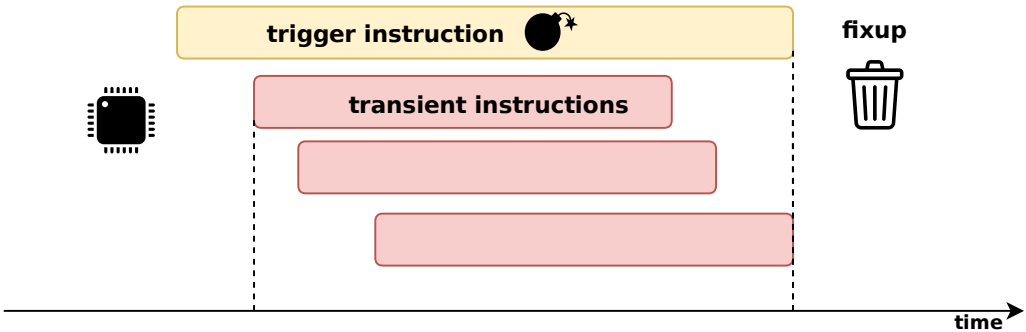



WHAT IF I TOLD YOU
YOU CAN CHANGE RULES MID-GAME
imgflip.com

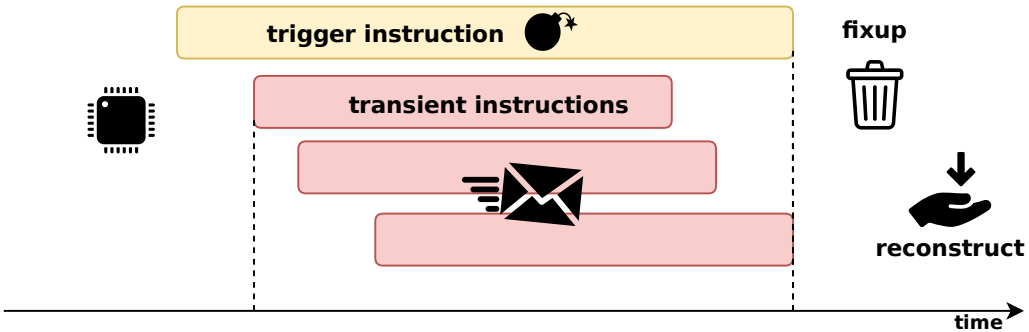
Abusing out-of-order and speculative execution



Abusing out-of-order and speculative execution



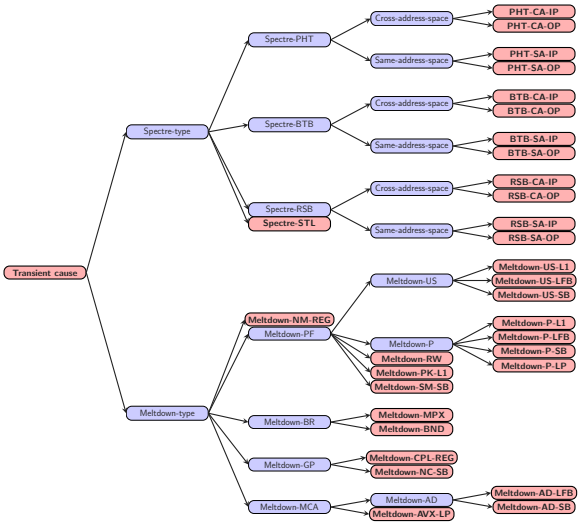
Abusing out-of-order and speculative execution



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

The transient-execution zoo

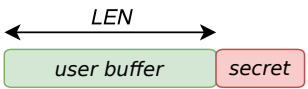
https://transient.fail



Attack vector 2: Transient execution

Idea #1: Confused-deputy code gadgets?

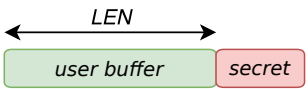
Spectre v1: Speculative buffer over-read




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

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

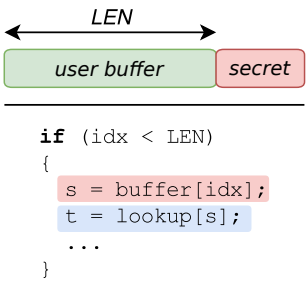
Spectre v1: Speculative buffer over-read




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

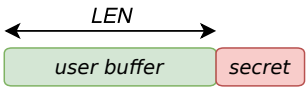
- Programmer *intention:* no out-of-bounds accesses
- **Mistrain gadget** to speculatively "ahead of time" execute with $idx \geq LEN$ in the transient world

Spectre v1: Speculative buffer over-read



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

Spectre v1: Speculative buffer over-read




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

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

Spectre take-away

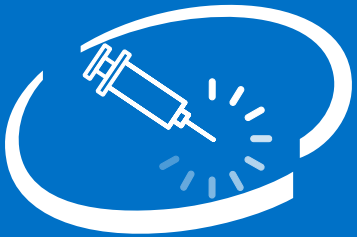
- CPU **transiently** executes wrong code paths
- **Confused-deputy gadgets** encode secrets via side channels

Attack vector 2: Transient execution

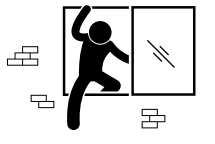
Idea #2: Transient access-control bypass?


inside™


inside™


inside™

Meltdown: Transiently encoding unauthorized memory



Unauthorized access

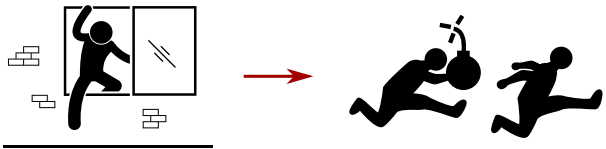
Listing 1: x86 assembly

```
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

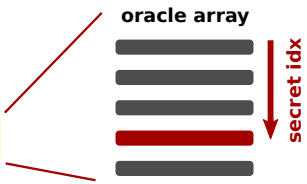
Transient out-of-order window

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 }
```



Meltdown: Transiently encoding unauthorized memory

Unauthorized access

Transient out-of-order window

Exception

(discard architectural state)

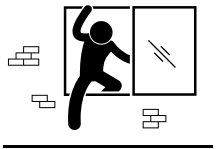
Listing 1: x86 assembly.

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

Listing 2: C code.

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

Meltdown: Transiently encoding unauthorized memory



Unauthorized access

Transient out-of-order window

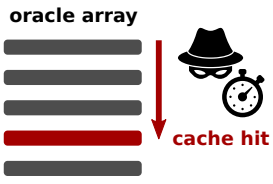
Exception handler

Listing 1: x86 assembly.

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

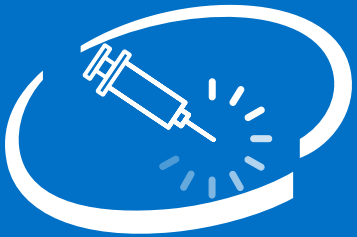
Listing 2: C code.

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



inside™

inside™



inside™

https://foreshadowattack.eu/

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

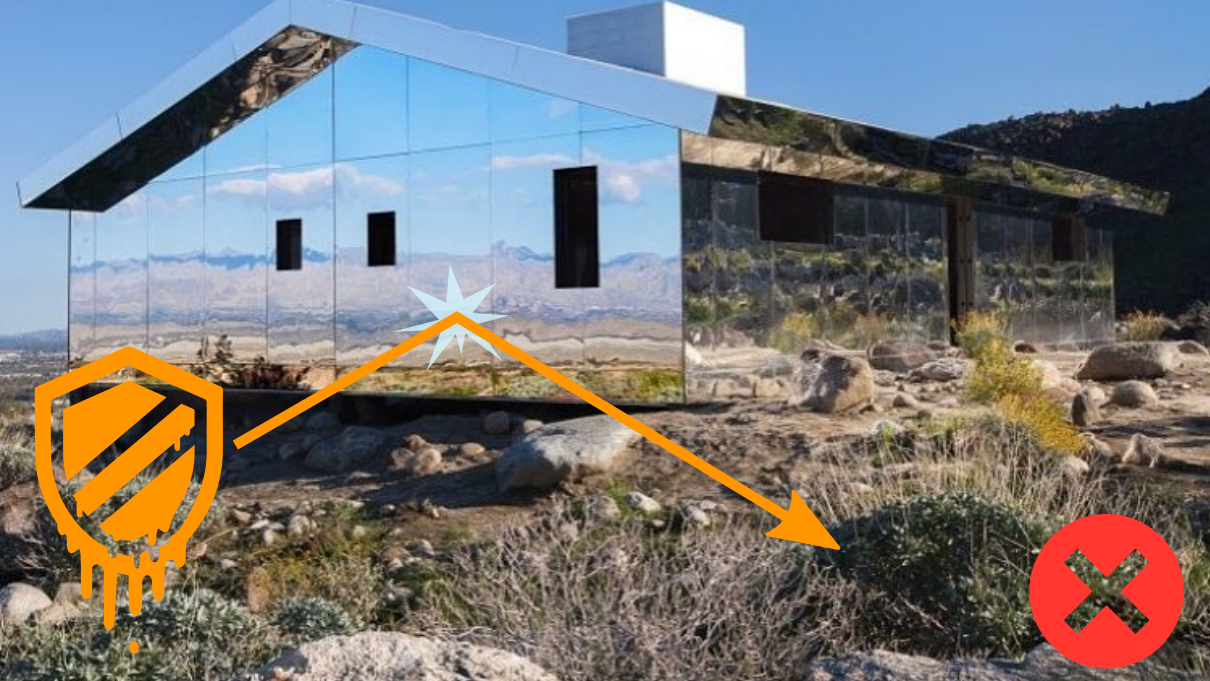
SPECTRE-LIKE FLAW UNDERMINES INTEL PROCESSORS' MOST SECURE ELEMENT

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

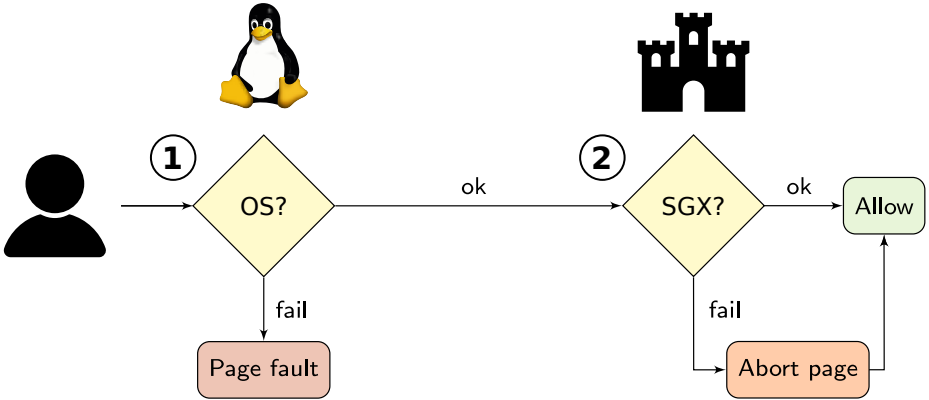
Intel's SGX blown wide open by, you guessed it, a speculative execution attack

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

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

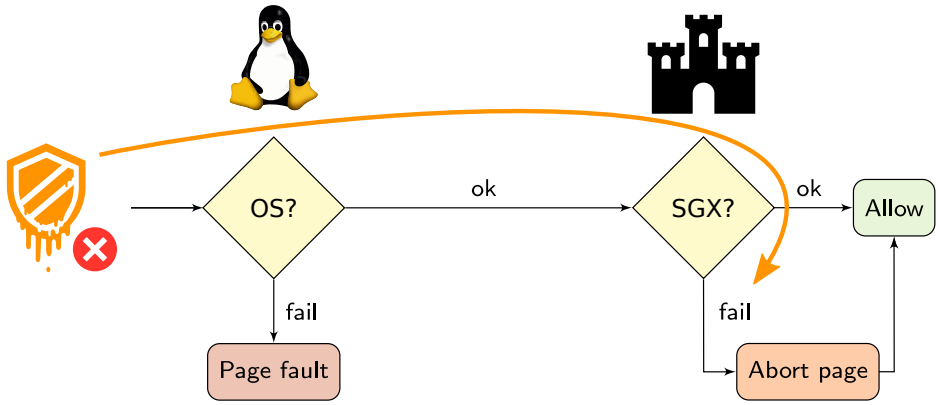


Building Foreshadow: Evade SGX abort page semantics



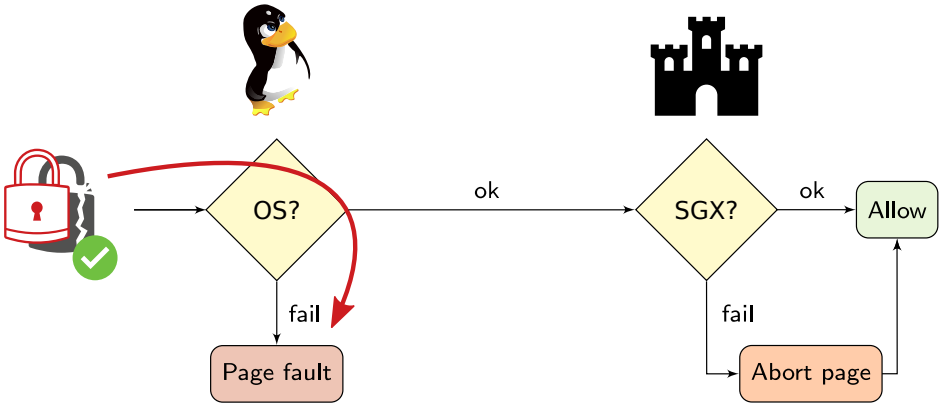
SGX checks prohibit unauthorized access

Building Foreshadow: Evade SGX abort page semantics



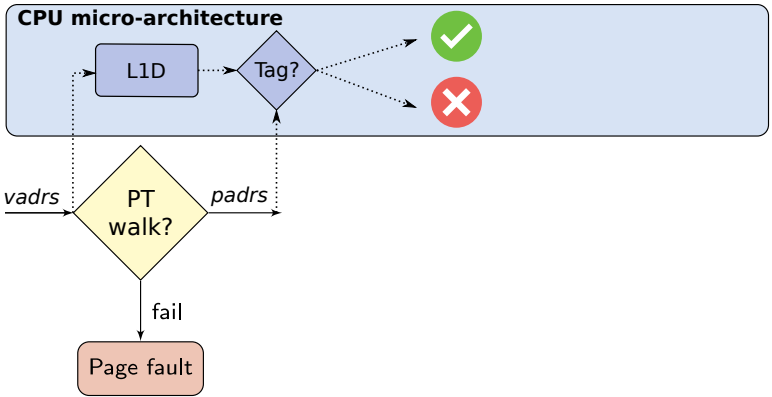
SGX checks prohibit unauthorized access

Building Foreshadow: Evade SGX abort page semantics



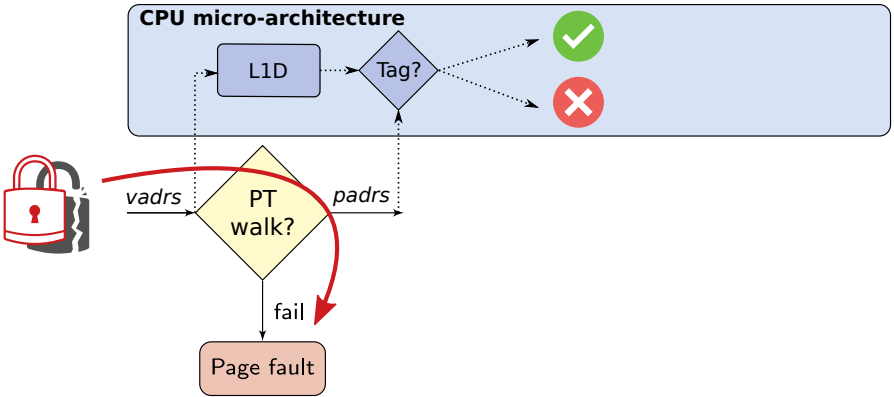
. . . but attackers can **unmap** enclave pages!

The microarchitecture behind Foreshadow



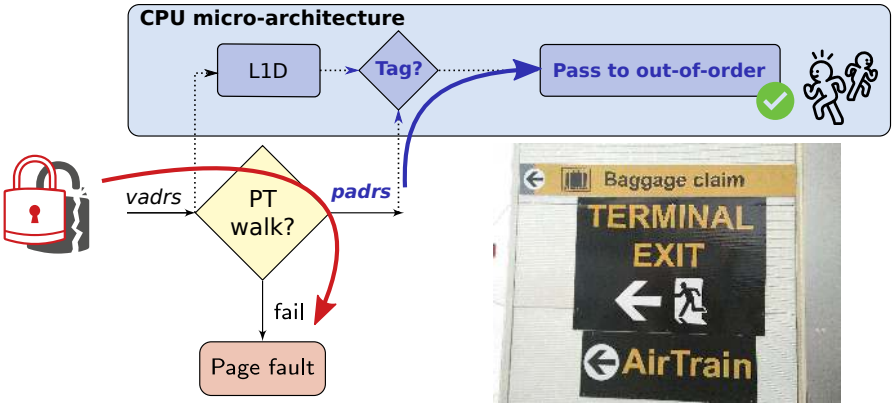
L1 cache design: Virtually-indexed, physically-tagged

The microarchitecture behind Foreshadow



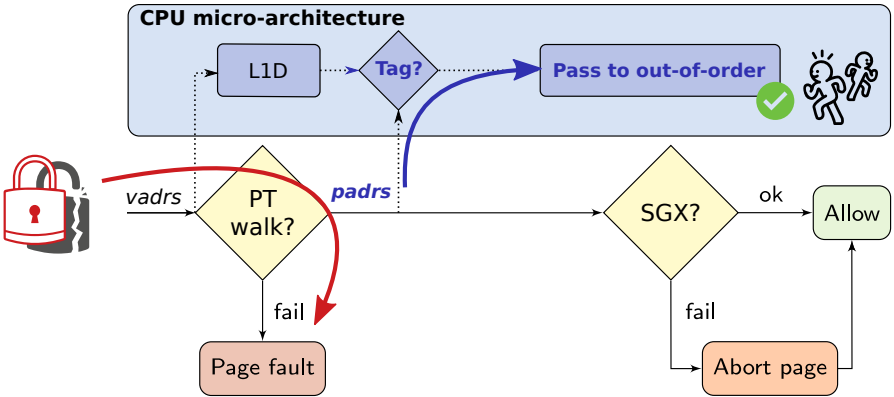
Page fault: Early-out address translation

The microarchitecture behind Foreshadow



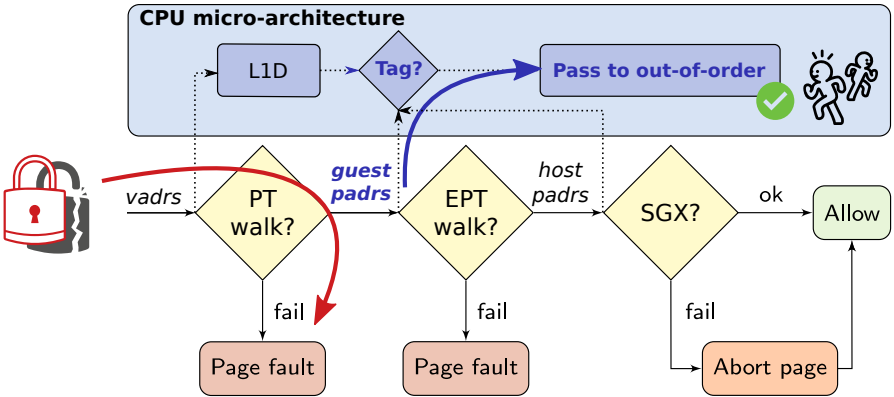
L1-Terminal Fault: Match *unmapped physical address* (!)

The microarchitecture behind Foreshadow

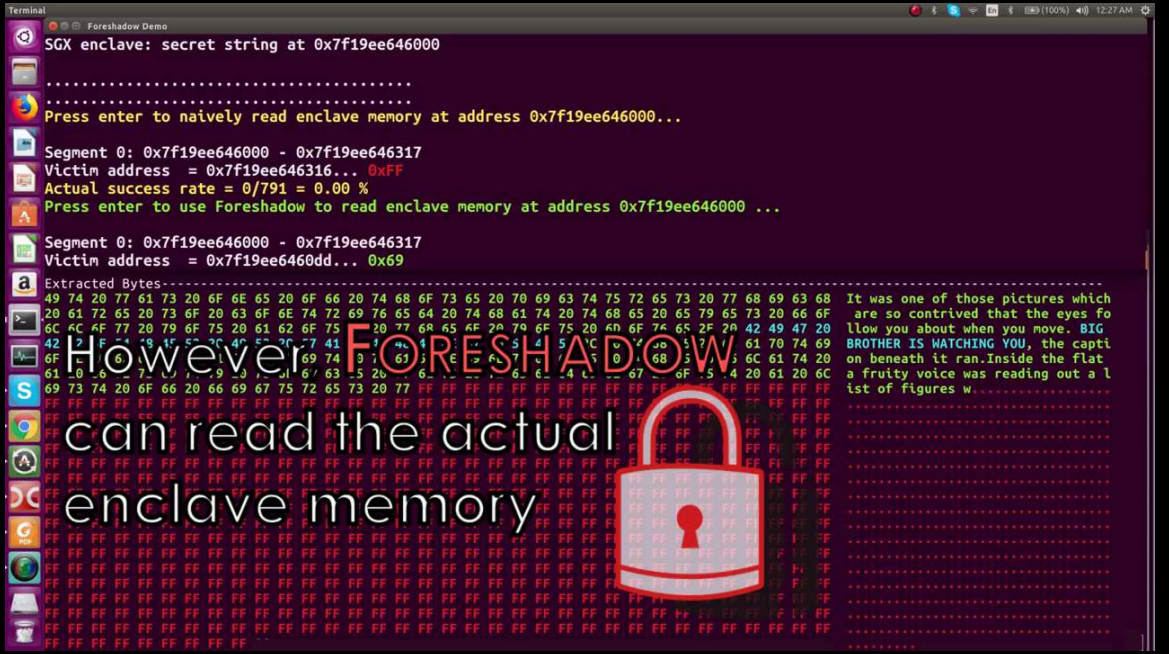


Foreshadow-SGX: Bypass enclave isolation

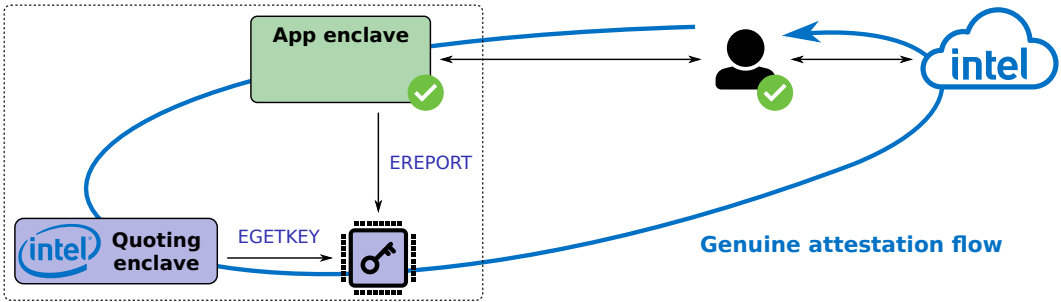
The microarchitecture behind Foreshadow



Foreshadow-NG: Bypass virtual machine isolation

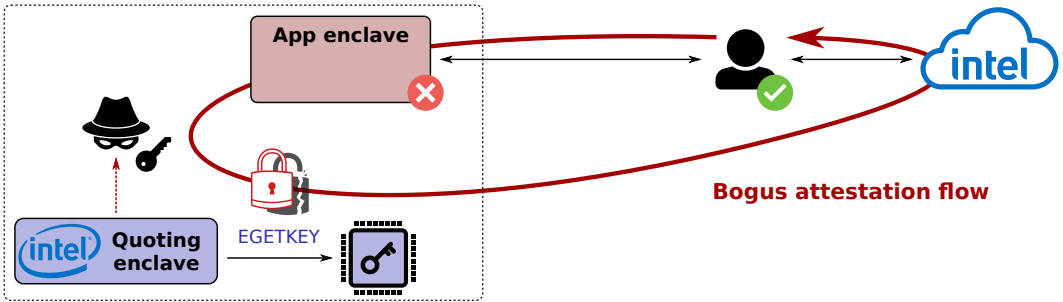

Terminal
Foreshadow Demo
SGX enclave: secret string at 0x7f19ee646000
Press enter to naively read enclave memory at address 0x7f19ee646000...
Segment 0: 0x7f19ee646000 - 0x7f19ee646317
Victim address = 0x7f19ee646316... 0xFF
Actual success rate = 0/791 = 0.00 %
Press enter to use Foreshadow to read enclave memory at address 0x7f19ee646000 ...
Segment 0: 0x7f19ee646000 - 0x7f19ee646317
Victim address = 0x7f19ee6460dd... 0x69
Extracted Bytes
However FORESHADOW
can read the actual
enclave memory

Foreshadow: Extracting the keys to the Intel SGX kingdom



Intel == trusted 3th party (shared **CPU master secret**)

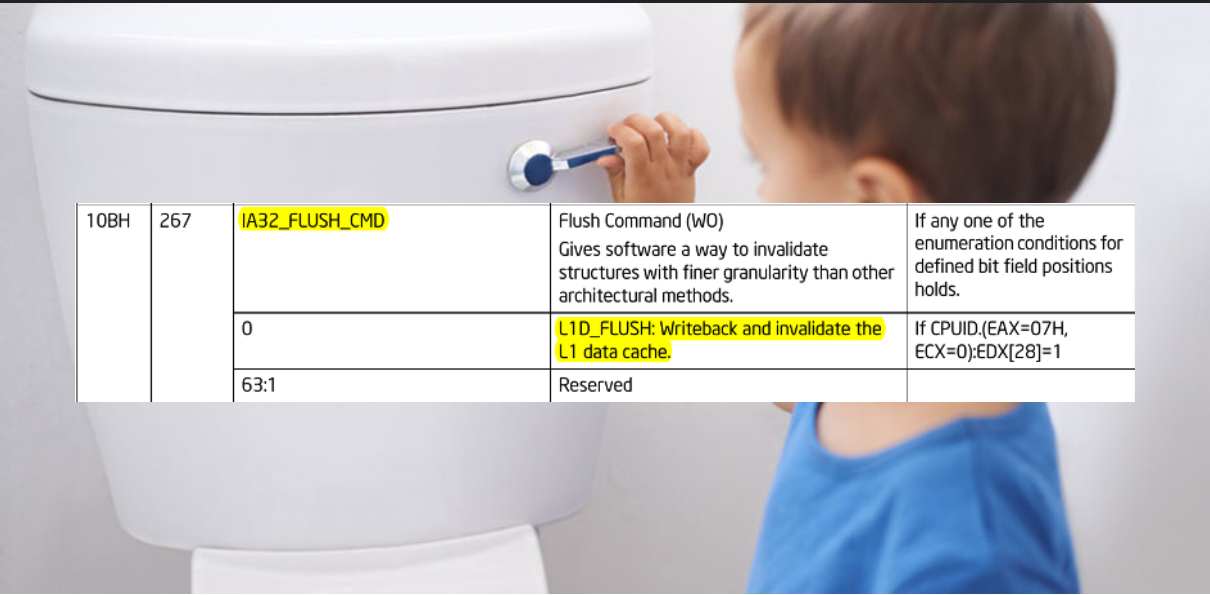
Foreshadow: Extracting the keys to the Intel SGX kingdom



Extract long-term platform **attestation key** → forge Intel signatures

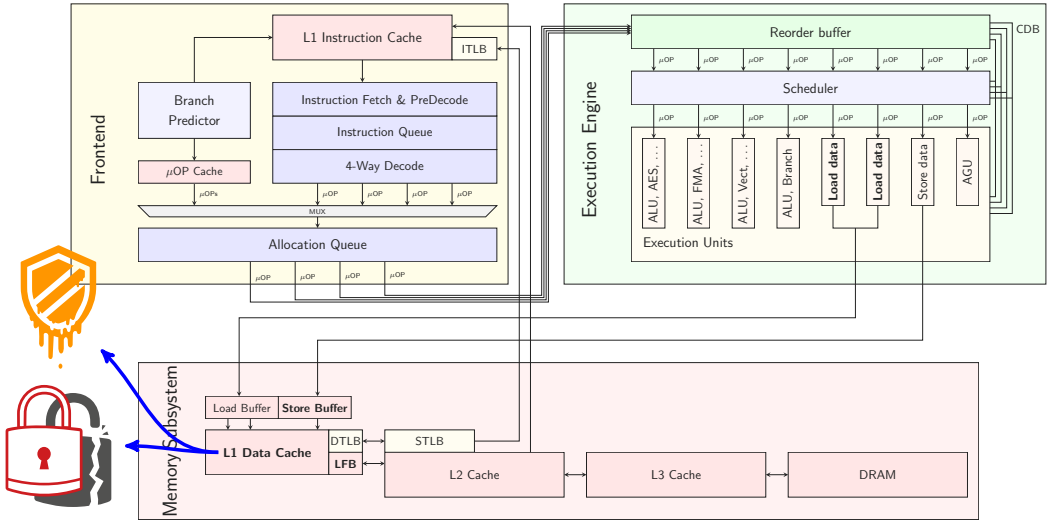
Mitigating Foreshadow: Flush CPU microarchitecture

Mitigating Foreshadow: Flush CPU microarchitecture

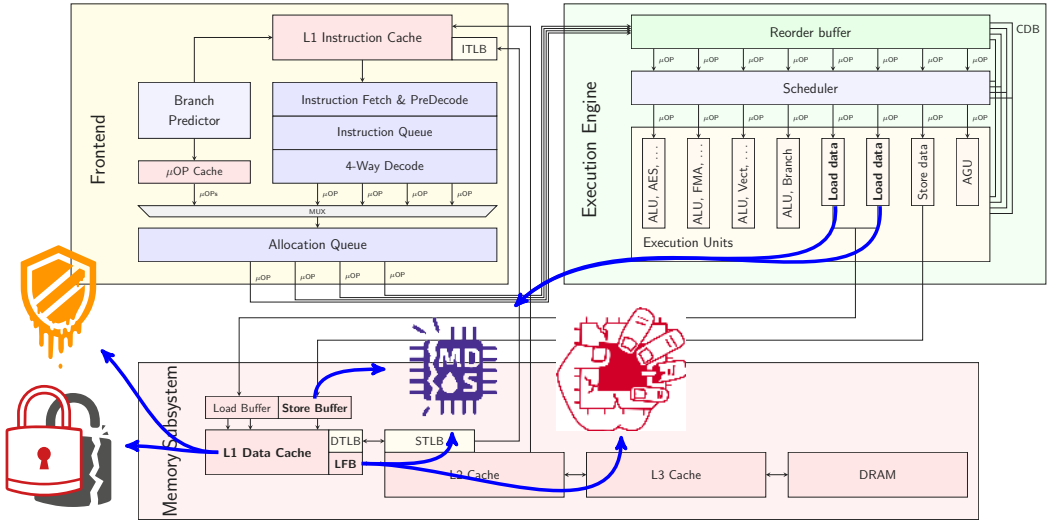


10BH	267	IA32_FLUSH_CMD	Flush Command (WO) Gives software a way to invalidate structures with finer granularity than other architectural methods.	If any one of the enumeration conditions for defined bit field positions holds.
		0	L1D_FLUSH: Writeback and invalidate the L1 data cache.	If CPUID.(EAX=07H,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

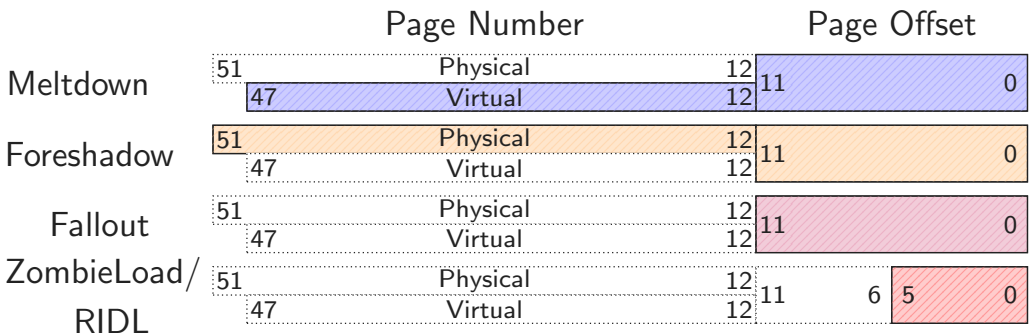
MDS variants: Flushing additional microarchitectural buffers



MDS variants: Flushing additional microarchitectural buffers



MDS variants: Address dependencies



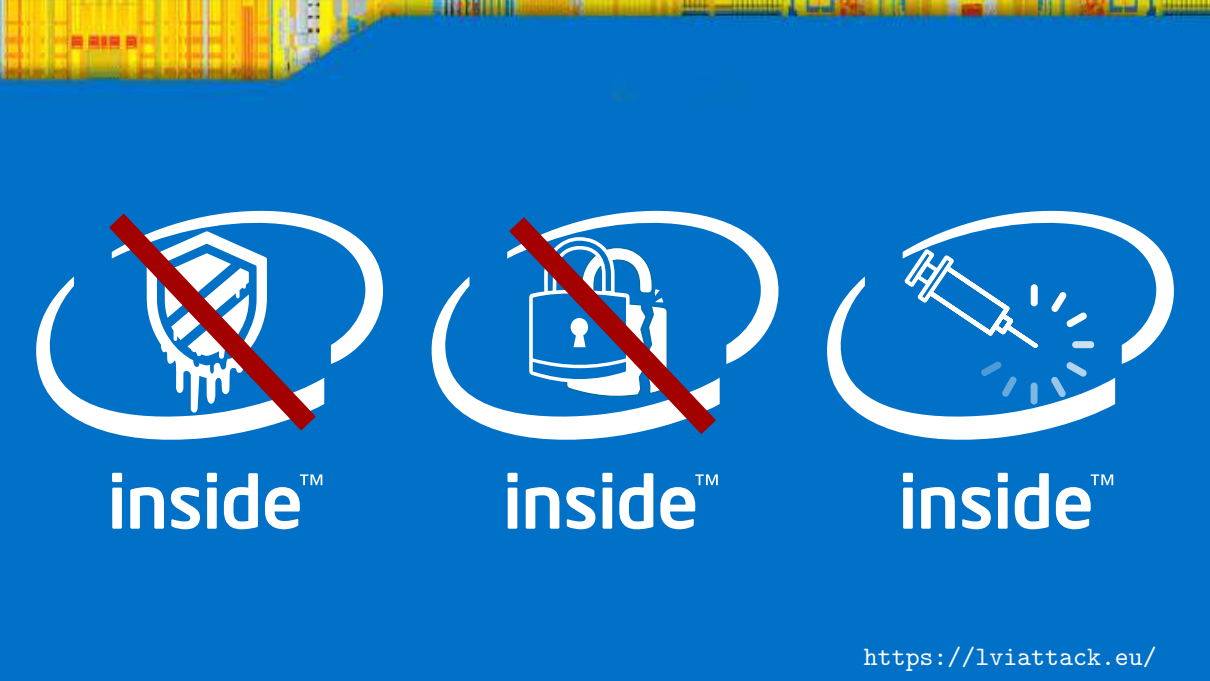
Take-away: Addressability ≠ (transient) accessibility

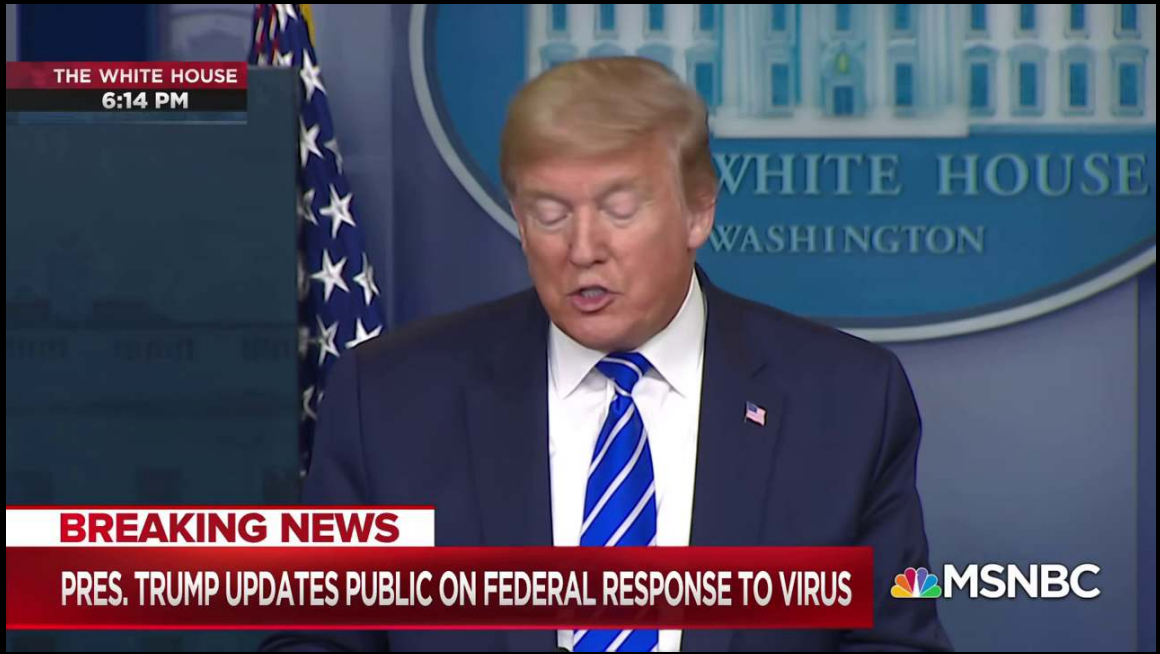
Meltdown take-away

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

Attack vector 2: Transient execution

Idea #3: Turn around transient data leakage?


inside™
inside™
inside™
https://lviattack.eu/


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

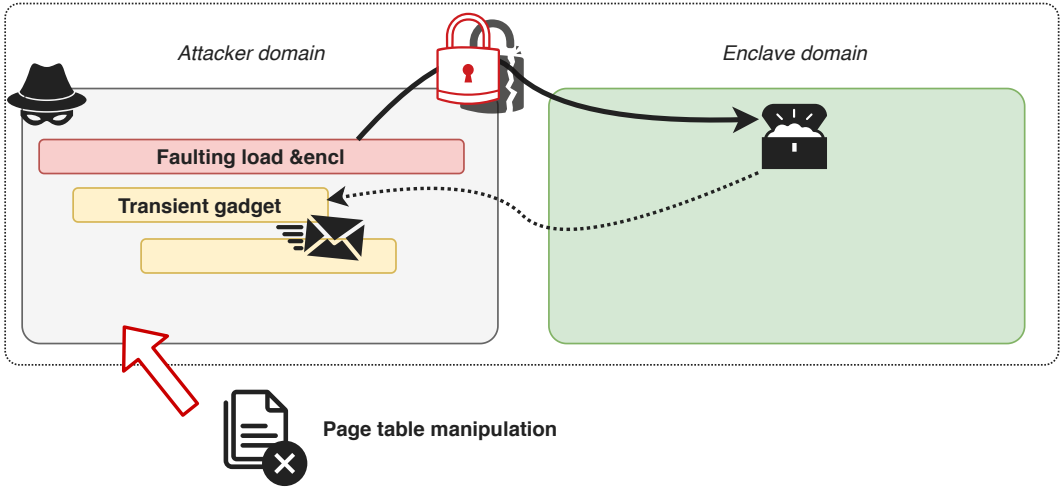
- Meltdown: out-of-reach
- Foreshadow: cache emptied

Intra-enclave view

- Access enclave + outside memory

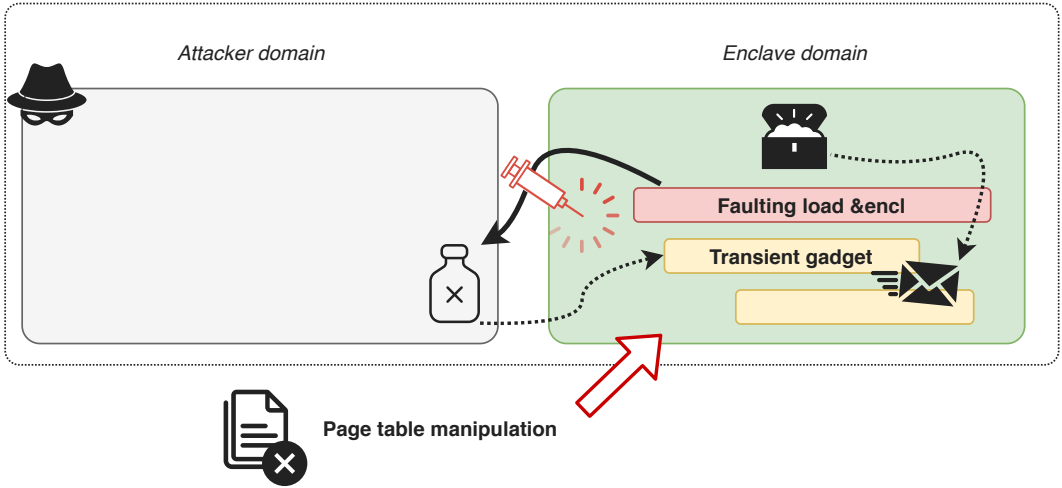
→ Abuse **in-enclave code gadgets!**

Reviving Foreshadow with Load Value Injection (LVI)



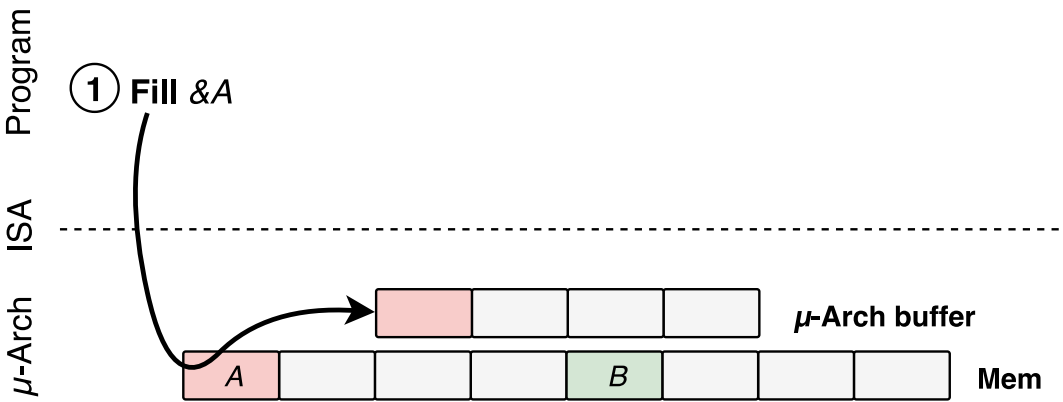
Van Bulck et al. "Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution", USENIX 2018.

Reviving Foreshadow with Load Value Injection (LVI)

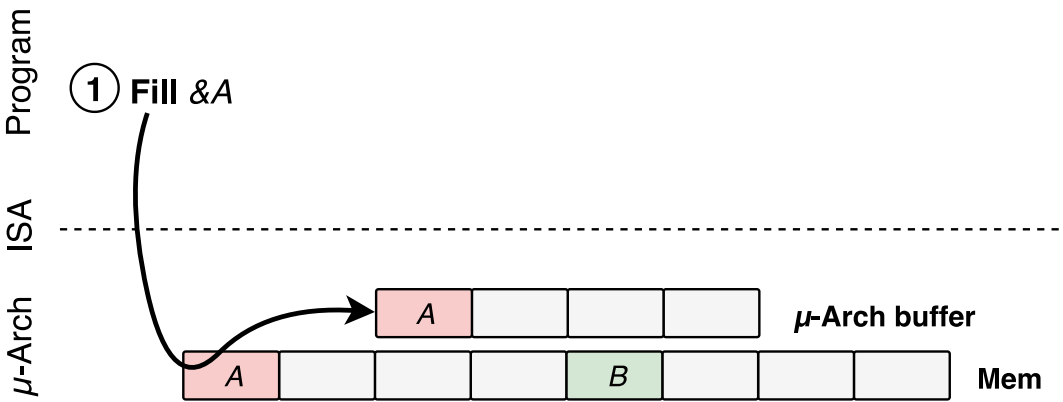


Van Bulck et al. "LVI: Hijacking Transient Execution through Microarchitectural Load Value Injection", S&P 2020.

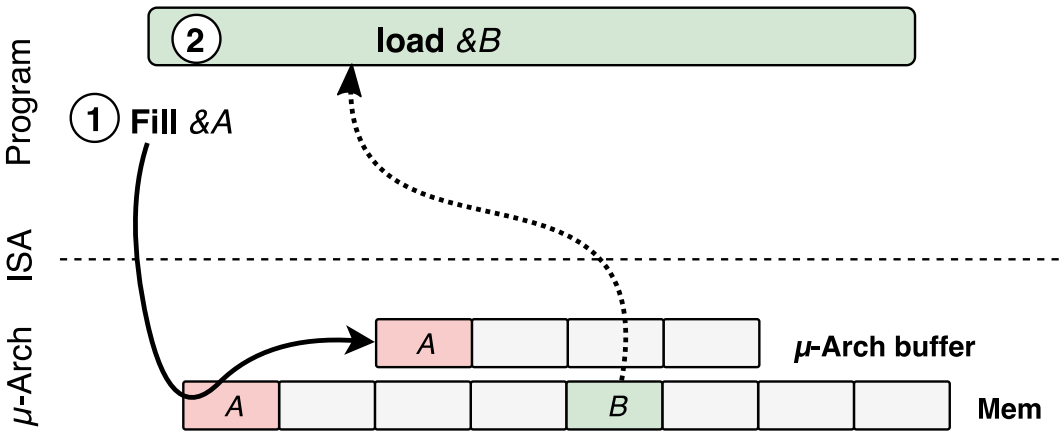
LVI: The basic idea



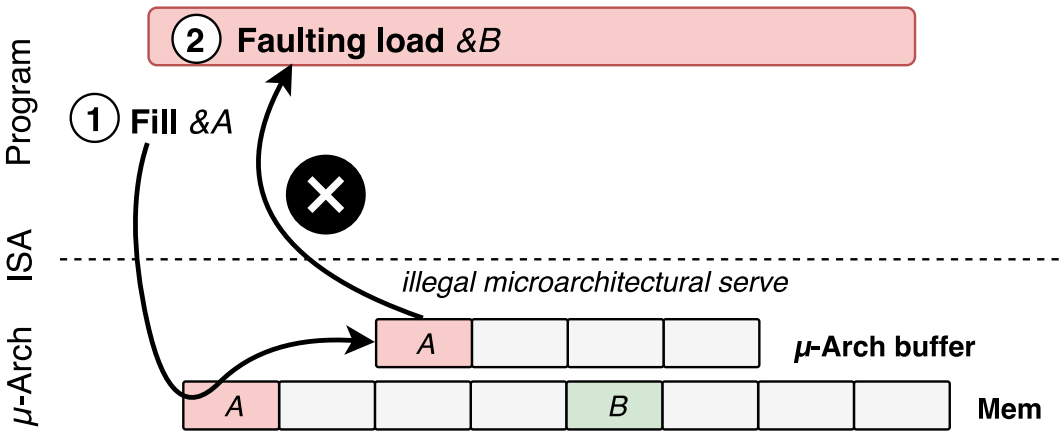
LVI: The basic idea



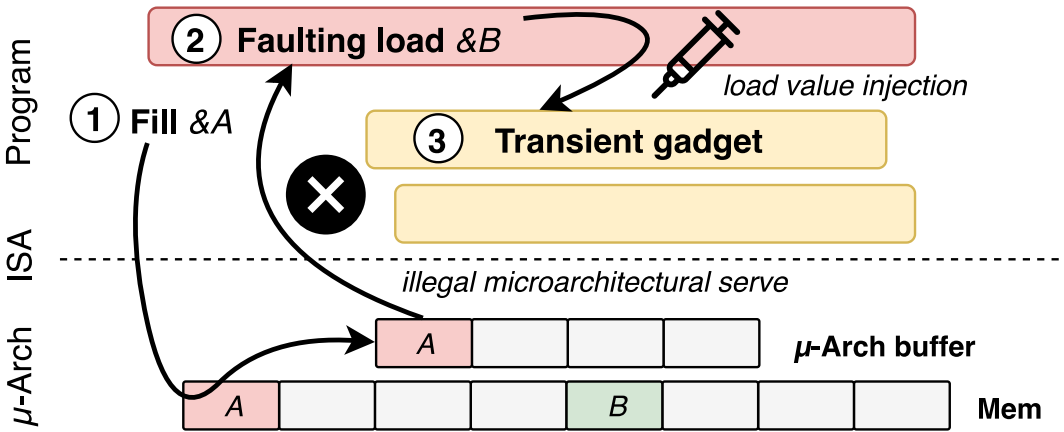
LVI: The basic idea



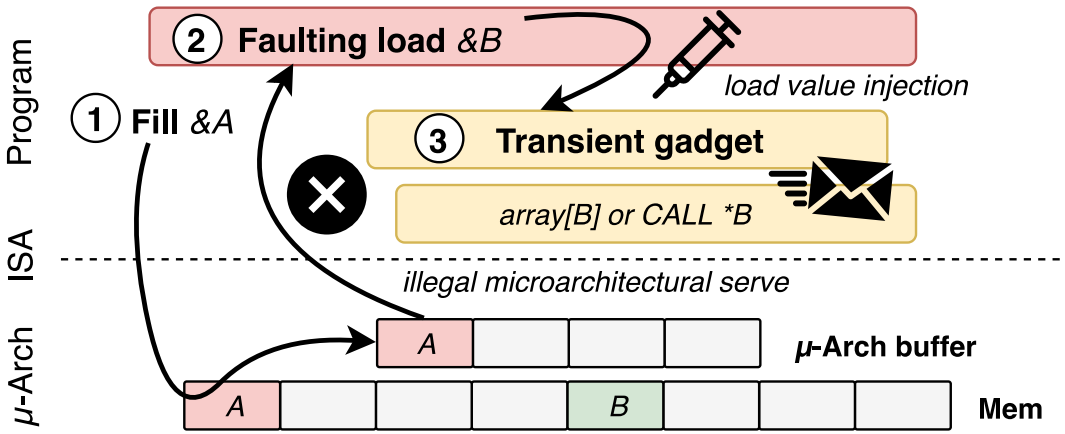
LVI: The basic idea



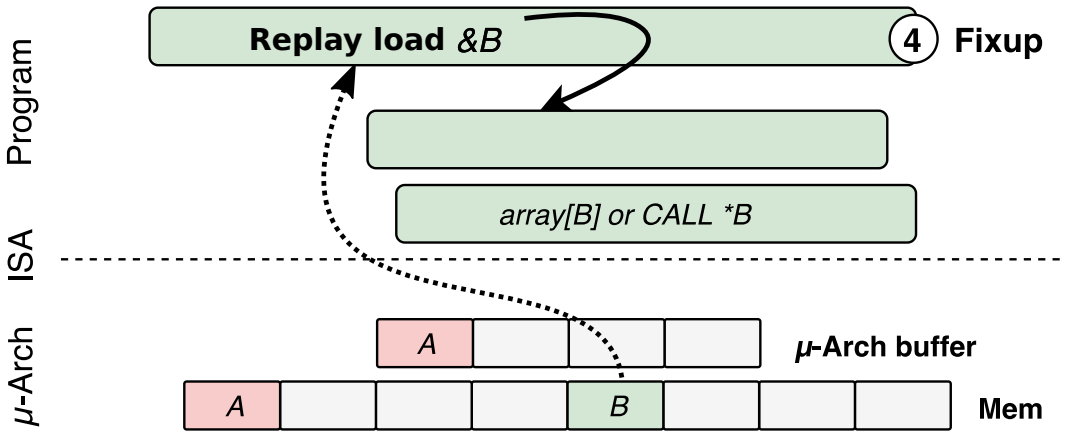
LVI: The basic idea



LVI: The basic idea



LVI: The basic idea

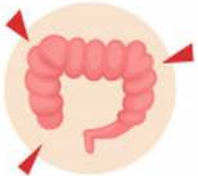


FOOD POISONING

Overdue products

Medicine

Dizziness

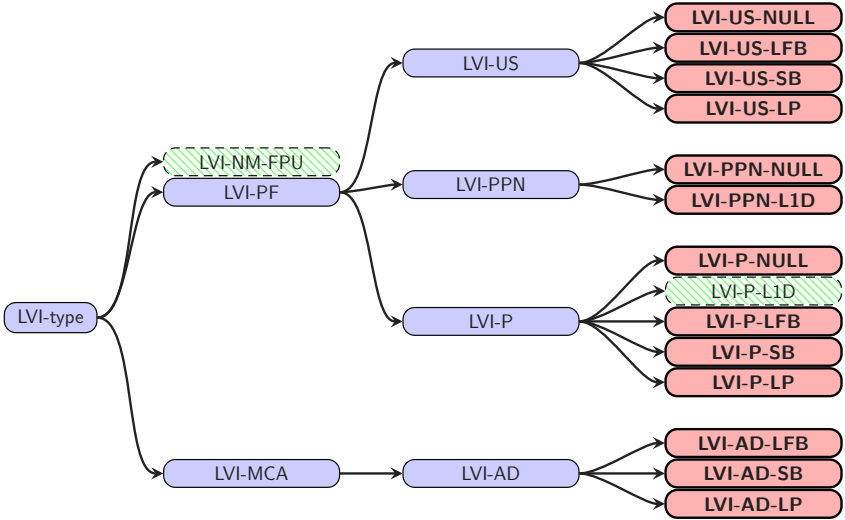


Intestinal colic

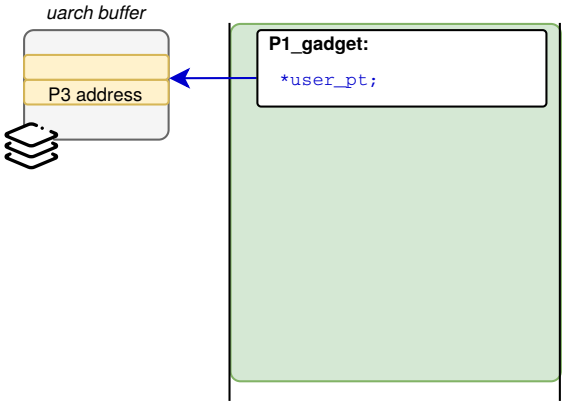
Diarrhea

Headache

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

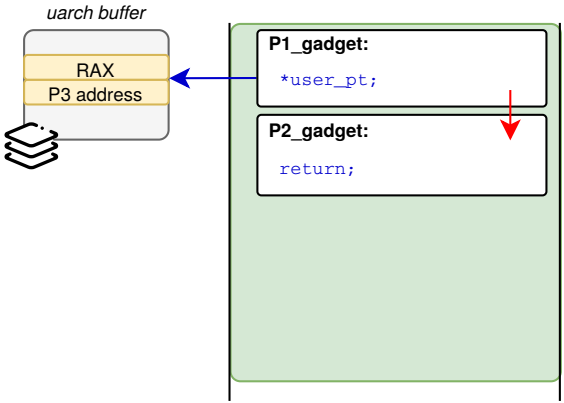


LVI-based transient control-flow hijacking



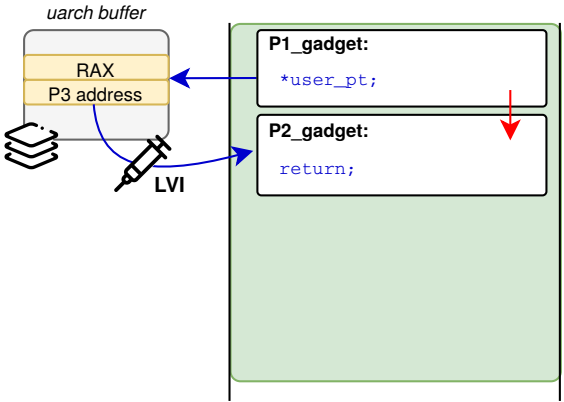
1. Victim fills μ-arch buffer with attacker-controlled data

LVI-based transient control-flow hijacking



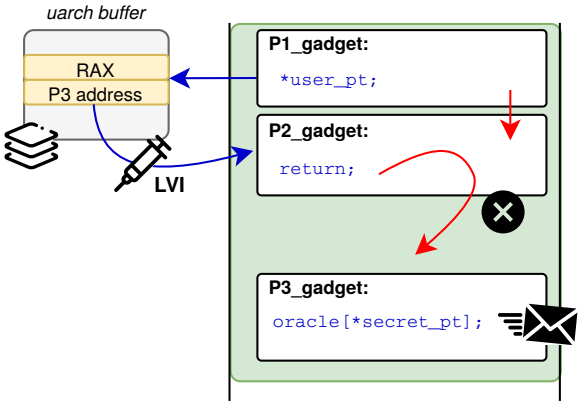
1. Victim fills μ-arch buffer with attacker-controlled data
2. Victim executes indirect branch (`JMP/CALL/RET`)

LVI-based transient control-flow hijacking



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

LVI-based transient control-flow hijacking

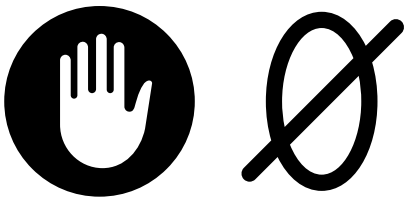


1. Victim fills μ-arch buffer with attacker-controlled data
2. Victim executes indirect branch (`JMP/CALL/RET`)
3. Faulting load → inject incorrect attacker values(!)
4. Redirect transient control flow

E/asm.S main.c

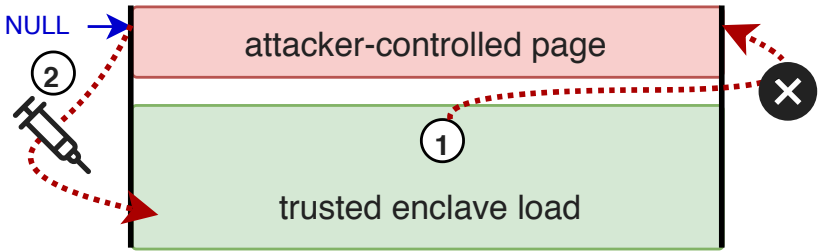
```
28      .global ecall_lvi_sb_rop
29      # %rdi store_pt
30      # %rsi oracle_pt
31  ecall_lvi_sb_rop:
32      mov %rsp, rsp_backup(%rip)
33      lea page_b(%rip), %rsp
34      add $OFFSET, %rsp
35
36      /* transient delay */
37      clflush dummy(%rip)
38      mov dummy(%rip), %rax
39
40      /* 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
```

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



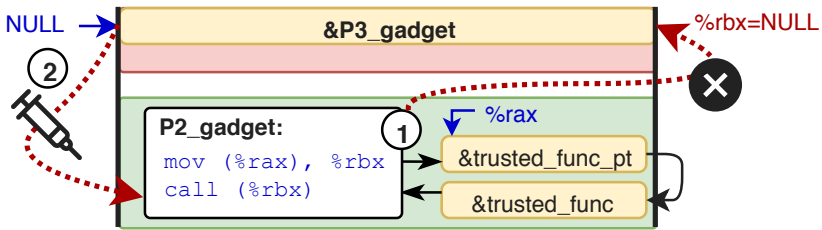
- Recent Intel CPUs forward 0x00 **dummy values** for faulting loads

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



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

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



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

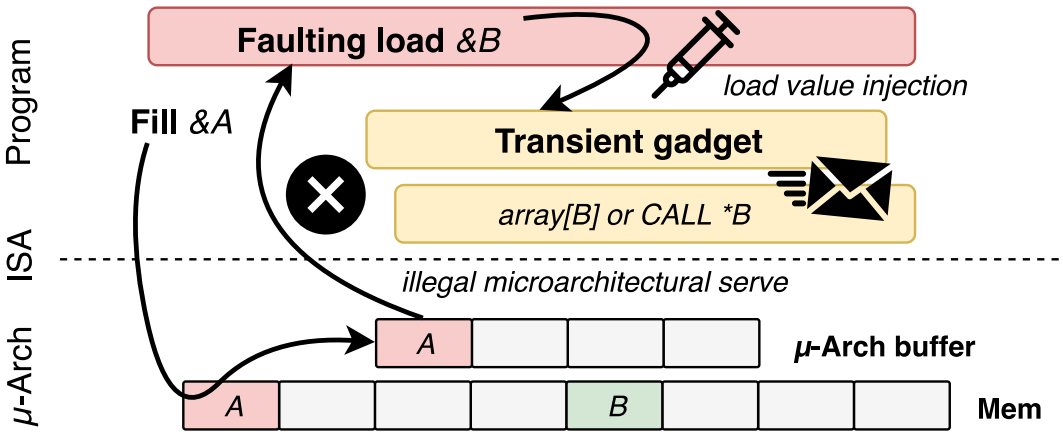
Mitigating LVI: Fencing vulnerable load instructions

Mitigating LVI: Fencing vulnerable load instructions

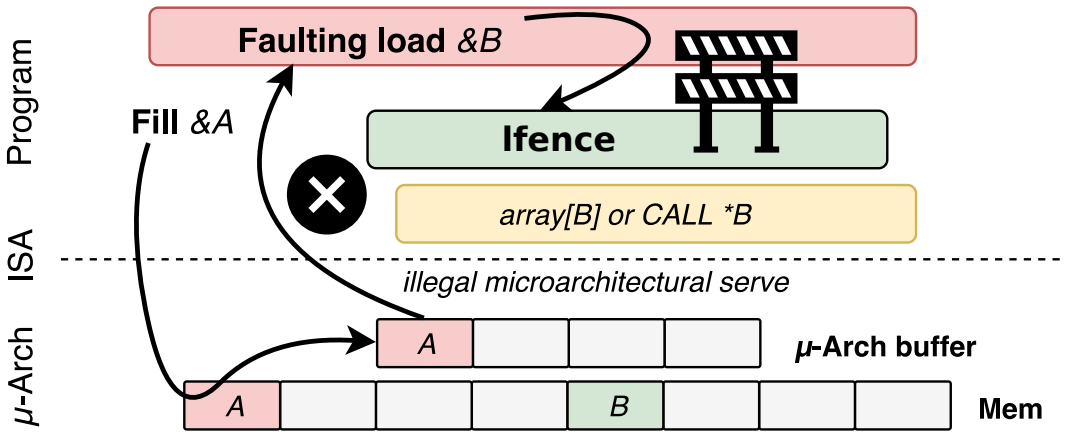
LFENCE—Load Fence

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
NP 0F AE E8	LFENCE	ZO	Valid	Valid	Serializes load operations.

Mitigation idea: Fencing vulnerable load instructions



Mitigation idea: Fencing vulnerable load instructions



Mitigating LVI: Compiler and assembler support

`-mlfence-after-load`

GNU Assembler Adds New Options For Mitigating Load Value Injection Attack

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

`-mlvi-hardening`

LLVM Lands Performance-Hitting Mitigation For Intel LVI Vulnerability

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

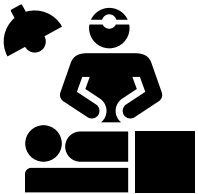
`-Qspectre-load`

More Spectre Mitigations in MSVC

March 13th, 2020

Intel architectural enclaves: `lfence` counts

`libsgx_qe.signed.so`

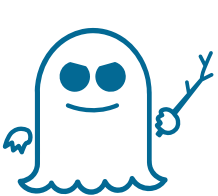


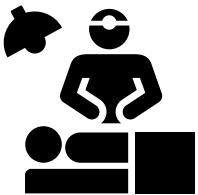
23 fences

October 2019—"surgical precision"

Intel architectural enclaves: `lfence` counts

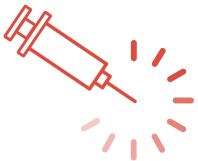
`libsgx_qe.signed.so`





23 fences

October 2019—"surgical precision"



49,315 fences

March 2020—"big hammer"

LVI mitigation performance overheads

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

LVI mitigation performance overheads

- **Short-term:** Expensive `lfence` barriers (or other compiler tricks, e.g., LVI-NULLify)
- **Mid-term:** Improved silicon patches in newer CPUs

Attack vector 3: CPU interfaces

Security perimeter for CPU-based isolation?

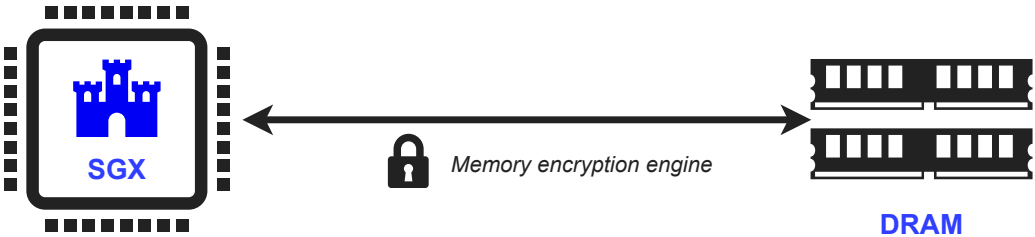
Security perimeter for CPU-based isolation?

SGX (wants to) only trust the **CPU package**
↔ CPU interfaces with the *physical world(!)*

Attack vector 3: CPU interfaces

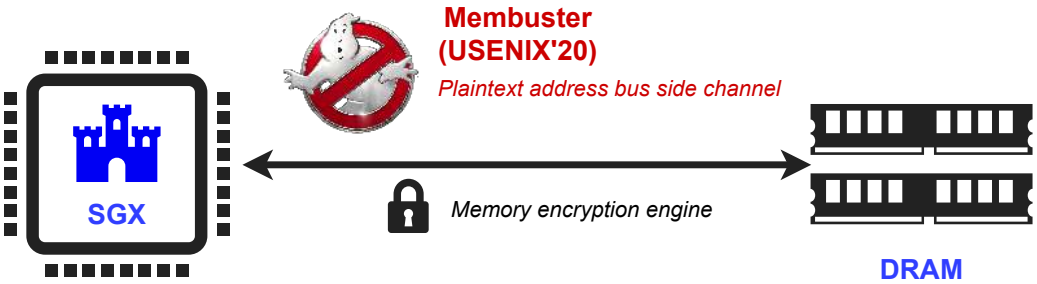
Idea #1: Memory bus?

Spying on the off-chip memory bus



Shay, "A Memory Encryption Engine Suitable for General Purpose Processors", 2016.

Spying on the off-chip memory bus



Lee et al. "An Off-Chip Attack on Hardware Enclaves via the Memory Bus", USENIX 20.

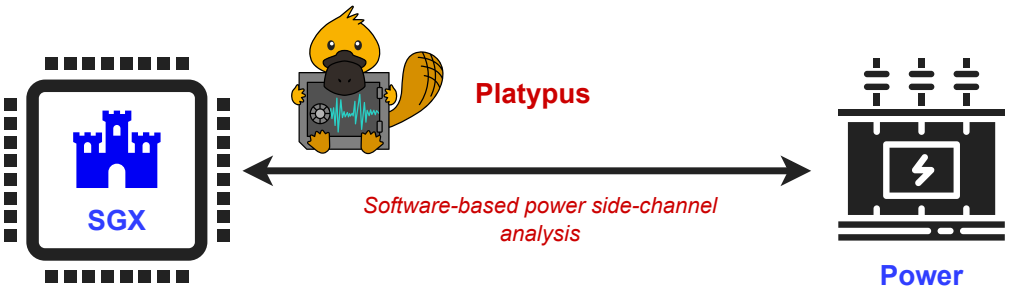
Requirements: Physical attacker → Side-channel *(metadata)* leakage

Attack vector 3: CPU interfaces

Idea #2: Power supply?

With great power comes great leakage ...

With great power comes great leakage . . .

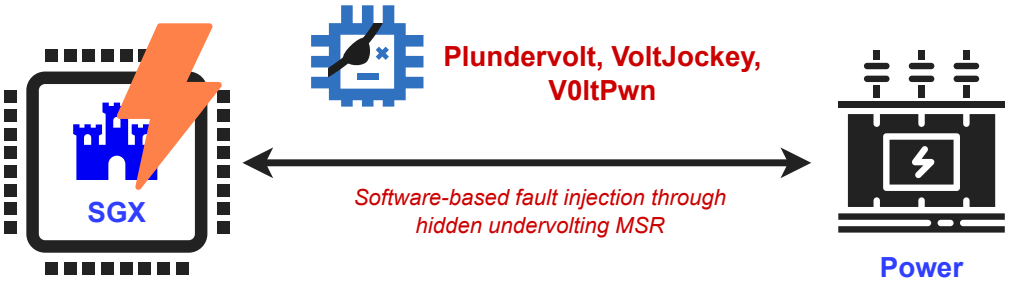


Lipp et al. "PLATYPUS: Software-based Power Side-Channel Attacks on x86", S&P 21.

Requirements: Software-only attacker → *Metadata* + *direct data* leakage(!)

How a little bit of undervolting can cause a lot of problems . . .

How a little bit of undervolting can cause a lot of problems . . .



Murdock et al. "Plundervolt: Software-Based Fault Injection Attacks against Intel SGX", S&P 20.

Requirements: Software-only attacker → Fault injection (*integrity* breach)

How a little bit of undervolting can cause a lot of problems . . .

Chen et al. "VoltPillager: Hardware-based fault injection attacks against Intel SGX Enclaves using the SVID voltage scaling interface", USENIX 21.

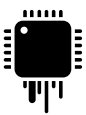
Requirements: Physical attacker → Fault injection (*integrity* breach)

Conclusions and takeaway

⇒ **Trusted execution** environments (Intel SGX) ≠ perfect(!)

⇒ Subtle **side channels** can go a long way...

⇒ **Privileged adversary model** = game changer

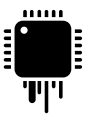


Conclusions and takeaway

⇒ **Trusted execution** environments (Intel SGX) ≠ perfect(!)

⇒ Subtle **side channels** can go a long way...

⇒ **Privileged adversary model** = game changer



Thank you! Questions?