Attacks and Defenses for Trusted Execution Environments

Jo Van Bulck

Seminar in Cybersecurity, KU Leuven – November 15 2024

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

DistriNet
KU LEUVEN

The Big Picture: Protecting Private Data

Data in transit

Data in use

Data at rest

The Big Picture: Protecting Private Data

Data in transit

- SSL/TLS etc.

Data in use

Data at rest

- Full disk encryption

The Big Picture: Protecting Private Data

Data in transit

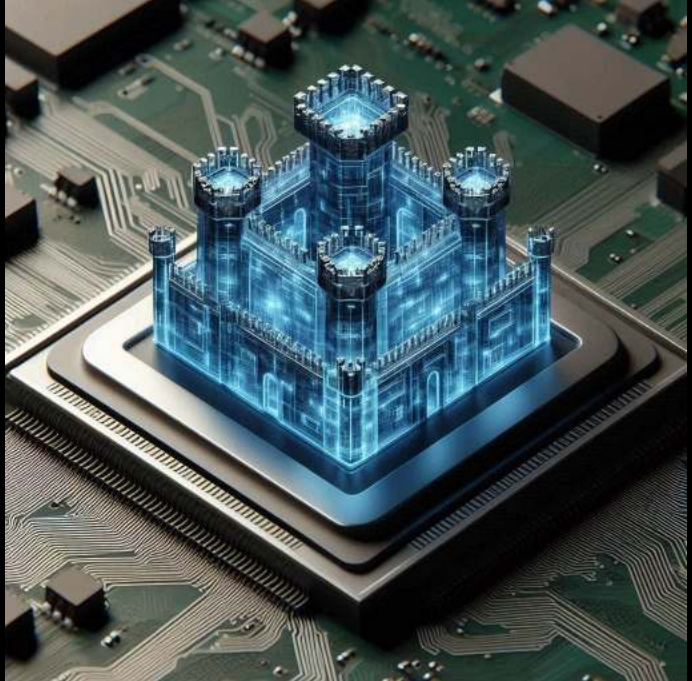
- SSL/TLS etc.

Data in use

- **Homomorphic encryption?**
- **Trusted Execution?**
  - = *Confidential Computing*
  - = *Hardware Enclaves*

Data at rest

- Full disk encryption



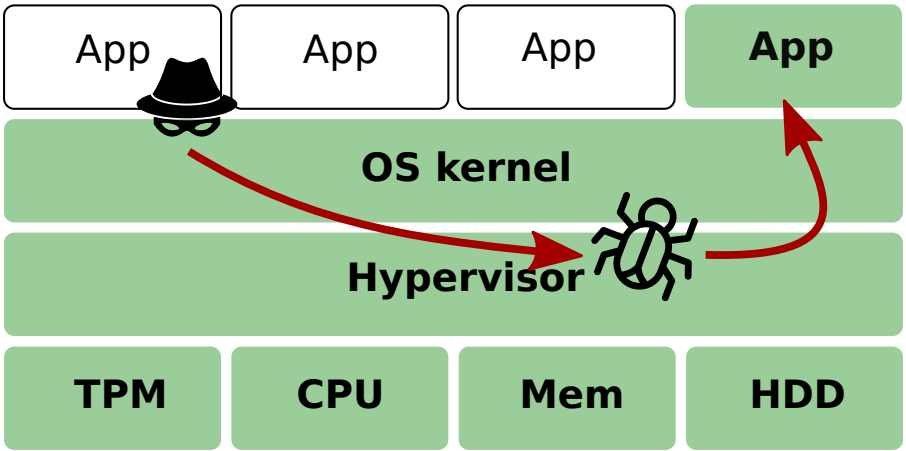
DALL·E 3

The Rise of Trusted Execution Environments

- 2004: ARM TrustZone
- 2015: **Intel Software Guard Extensions (SGX)**
- 2016: AMD Secure Encrypted Virtualization (SEV)
- 2018: IBM Protected Execution Facility (PEF)
- 2020: AMD SEV with Secure Nested Paging (SEV-SNP)
- 2022: Intel Trust Domain Extensions (TDX)
- 2023: ARM Confidential Compute Architecture (CCA)
- 2024: NVIDIA Confidential Computing

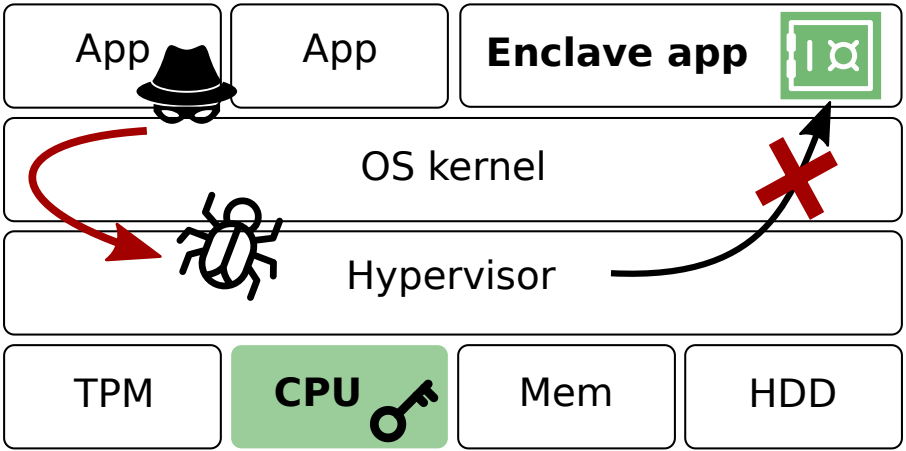
TEEs are here to stay...

Goal: Reducing Attack Surface with Enclaves



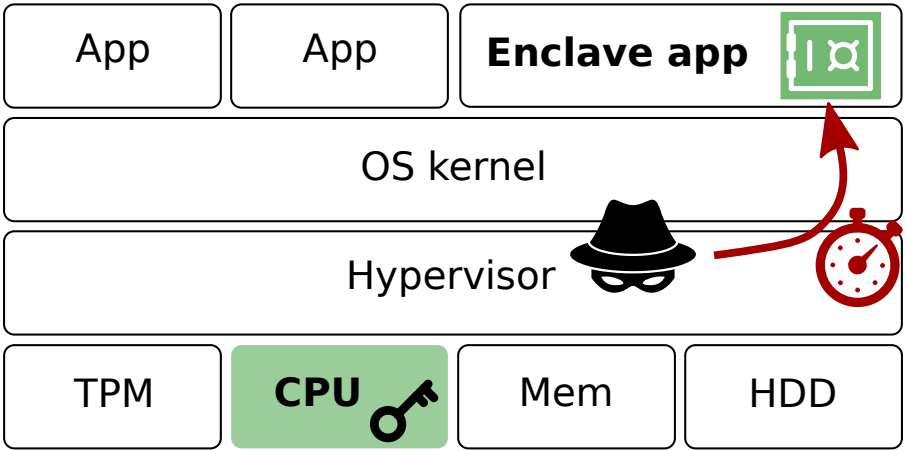
Traditional **layered designs:** Large trusted computing base

Goal: Reducing Attack Surface with Enclaves



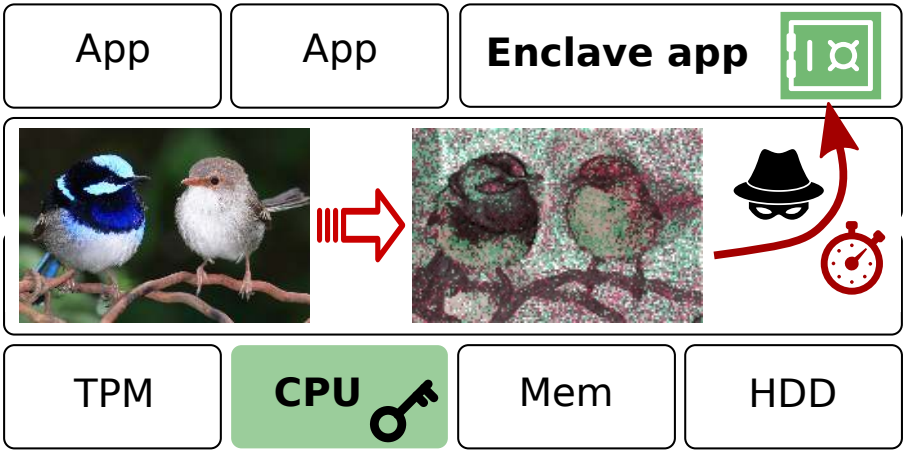
Intel SGX promise: Hardware-level **isolation and attestation**

Reality: Privileged Side-Channel Attacks



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

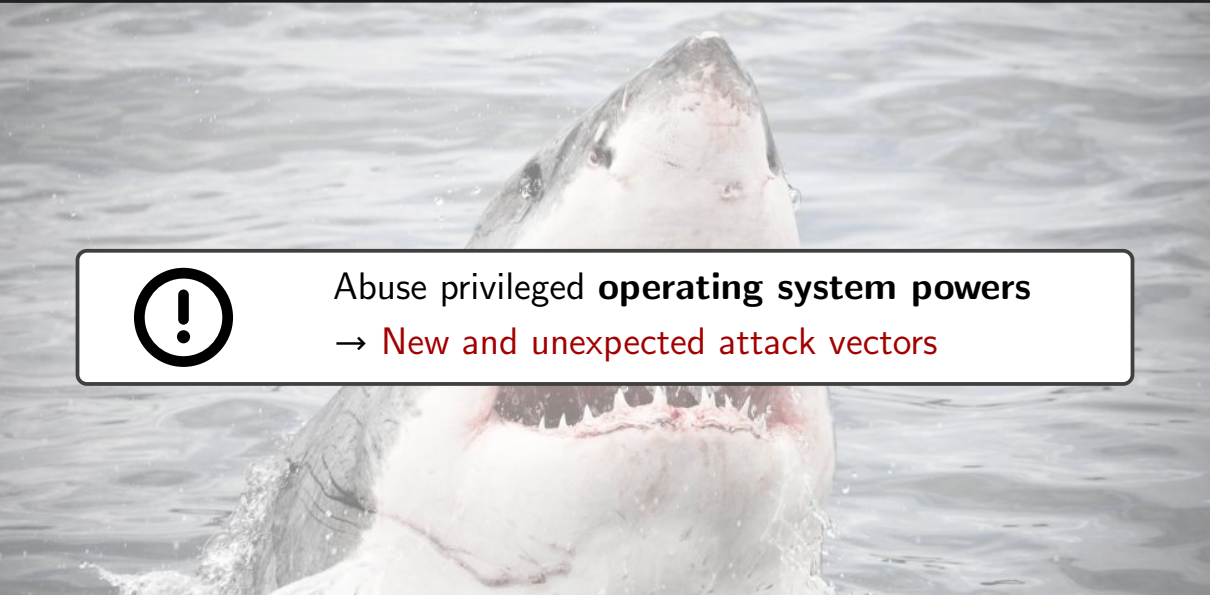
Reality: Privileged Side-Channel Attacks



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

Game Changer: Privileged “Bottom-Up” Adversary Model

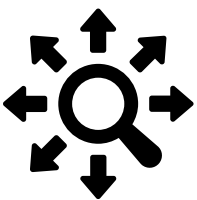
Game Changer: Privileged “Bottom-Up” Adversary Model



Abuse privileged **operating system powers**

→ New and unexpected attack vectors

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

Systematizing the SGX Attack Landscape

Metadata vs. direct data; confidentiality vs. integrity

	Attack \ Properties	x86 Interface						Constraints			
		PTE	GDT	IRQ	MSR	CR0	PMC	SMT	REP	SAS	Granular
μ-arch contention	Cache priming [181, 92, 29, 225, 79]	◐	○	◐	◐	○	◐	◐	●	○	64 B
	Branch prediction [156, 59, 105]	◐	○	◐	◐	◐	◐	◐	●	○	Inst
	DRAM row buffer conflicts [263]	○	○	○	◐	◐	○	○	●	○	1-8 KiB
	False dependencies [180]	○	○	○	◐	○	○	●	●	○	4 B
	Interrupt latency [**256**, 95, 208]	◐	○	●	◐	○	○	○	●	○	Inst
	Port contention [7]	○	○	○	◐	○	○	●	●	○	*μ*-op
Control channel	Page faults [277]	●	○	○	○	○	○	○	○	○	4 KiB
	Page table A/D [**258**, 263]	●	○	◐	○	◐	○	◐	○	○	4 KiB
	Page table flushing [**258**]	●	○	◐	○	○	○	◐	○	○	32 KiB
	Interrupt counting [**182**]	●	○	●	○	○	○	○	○	○	Inst
	IA32 segmentation faults [**91**]	◐	●	◐	○	○	○	○	○	○	1 B-4 KiB
	Alignment faults [**254**]	○	○	●	○	●	○	○	○	○	1 B
Transient	Foreshadow L1D extraction [**249**]	●	○	◐	○	○	○	◐	○	○	–
	Data sampling [**223**, 216, 211]	●	○	◐	○	○	○	◐	◐	○	–
	Spectre [39, 148]	◐	○	◐	○	○	○	◐	◐	◐	–
	Load value injection [**251**]	●	○	◐	○	○	○	◐	◐	◐	–
Interface	Memory safety [**254**, 154, 24, 266]	◐	○	◐	○	○	○	○	◐	◐	–
	Undervolting [**188**, 137, 210]	◐	○	◐	●	○	○	◐	●	◐	–
	Off-chip memory address bus [152]	●	○	◐	○	◐	○	○	○	○	64 B

https://vanbulck.net/files/phd-thesis.pdf

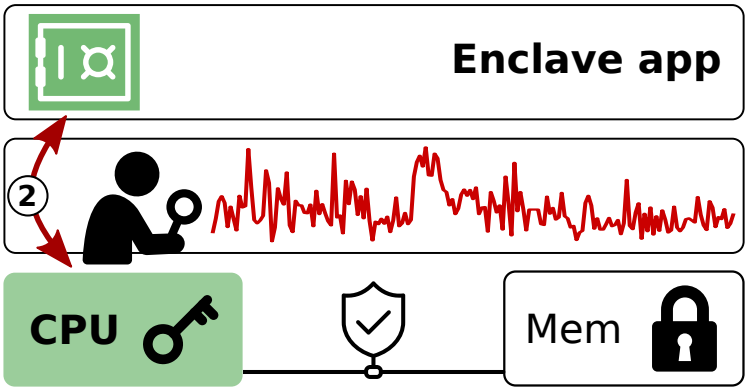
Systematizing the SGX Attack Landscape

Attack \ Properties		x86 Interface						Constraints			
		PTE	GDT	IRQ	MSR	CR0	PMC	SMT	REP	SAS	Granular
µ-arch contention	Cache priming [181, 92, 29, 225, 79]	◐	○	◐	◐	○	◐	◐	●	○	64 B
	Branch prediction [156, 59, 105]	◐	○	◐	◐	◐	◐	◐	●	○	Inst
	DRAM row buffer conflicts [263]	○	○	○	◐	◐	○	○	●	○	1-8 KiB
	False dependencies [180]	○	○	○	◐						
	Interrupt latency [**256**, 95, 208]	◐	○	●	◐						
	Port contention [7]	○	○	○	◐						
Control channel	Page faults [277]	●	○	○	○						
	Page table A/D [**258**, 263]	●	○	◐	○						
	Page table flushing [**258**]	●	○	◐	○						
	Interrupt counting [**182**]	●	○	●	○	○	○	○	○	○	Inst
	IA32 segmentation faults [**91**]	◐	●	◐	○	○	○	○	○	○	1 B-4 KiB
	Alignment faults [**254**]	○	○	●	○	●	○	○	○	○	1 B
Transient	Foreshadow L1D extraction [**249**]	●	○	◐	○	○	○	◐	○	○	–
	Data sampling [**223**, 216, 211]	●	○	◐	○	○	○	◐	◐	○	–
	Spectre [39, 148]	◐	○	◐	○	○	○	◐	◐	◐	–
	Load value injection [**251**]	●	○	◐	○	○	○	◐	◐	◐	–
Interface	Memory safety [**254**, 154, 24, 266]	◐	○	◐	○	○	○	○	◐	◐	–
	Undervolting [**188**, 137, 210]	◐	○	◐	●	○	○	◐	●	◐	–
	Off-chip memory address bus [152]	●	○	◐	○	◐	○	○	○	○	64 B

Possibly every privileged CPU feature can be abused!

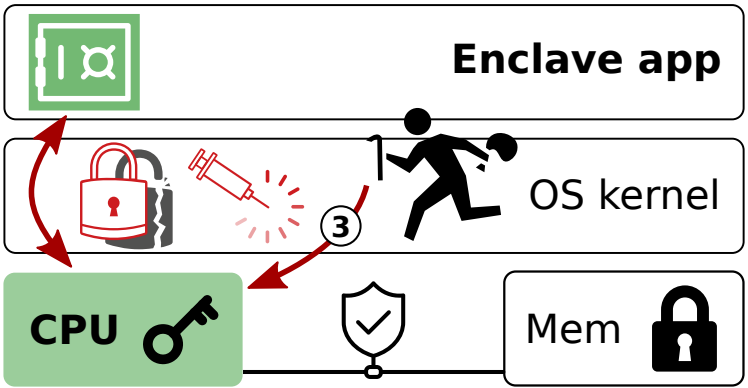
https://vanbulck.net/files/phd-thesis.pdf

Lecture Overview: Privileged Side-Channel Attacks (part 1)



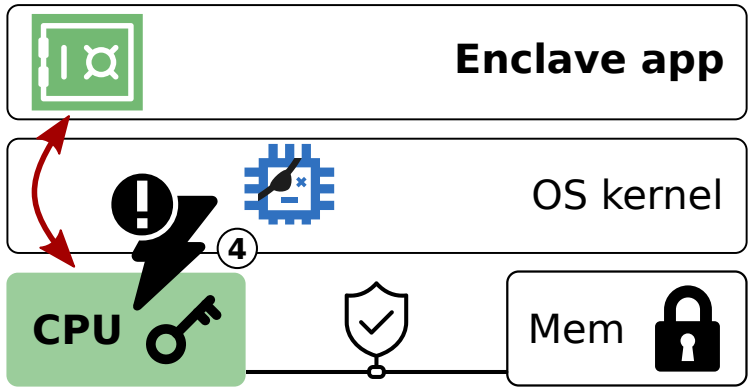
Privileged side channels to spy on enclave-CPU interaction metadata

Lecture Overview: Transient-Execution Attacks (part 2)



Transient-execution for direct data extraction from CPU

Lecture Overview: Privileged CPU Interface Attacks (part 3)



Unexpected (physical) **CPU interface** manipulations

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.

1. Privileged Side-Channel Attacks

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.

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 }
```

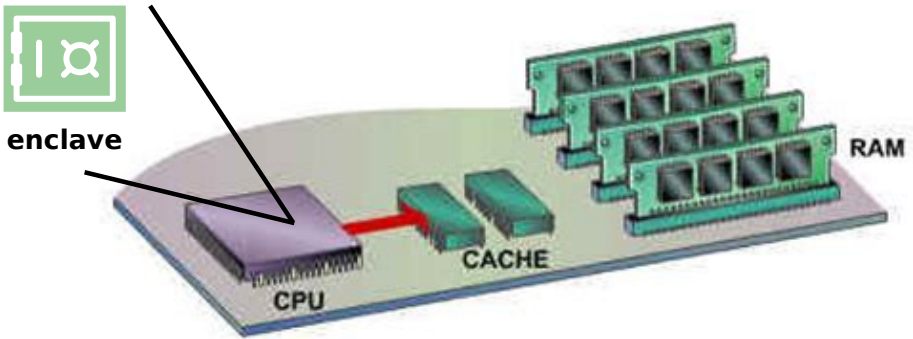
Vulnerable patterns: Secret-dependent code/data accesses

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

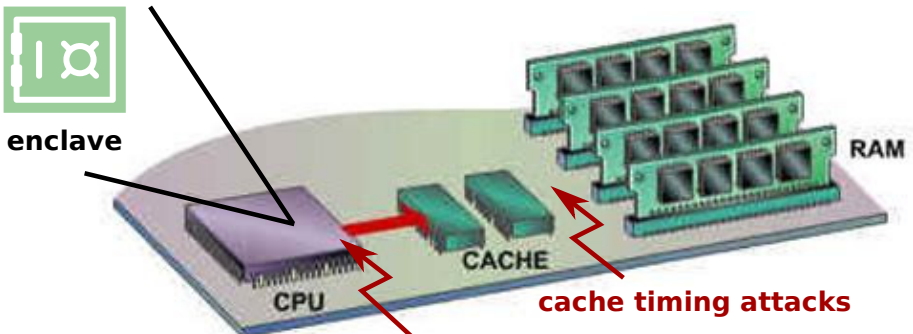
```
1 int secret_lookup(int s)
2 {
3     if (s > 0 && s < ARRAY_LEN)
4         return array[s];
5     return -1;
6
7 }
```

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

Overview: Spying on enclave memory accesses



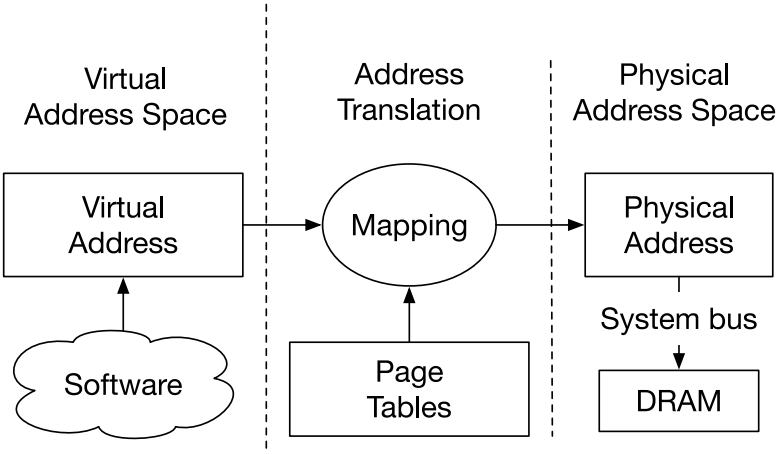
Overview: Spying on enclave memory accesses



1. Privileged Side-Channel Attacks

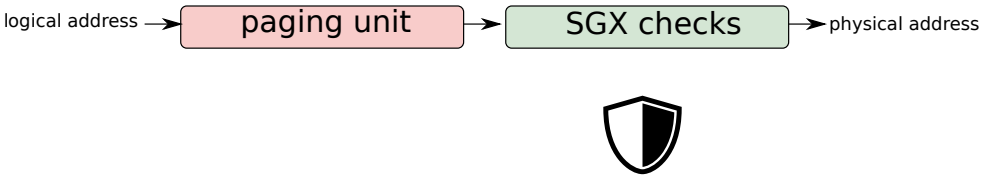
Idea #1: Monitoring Address Translation?

The virtual memory abstraction



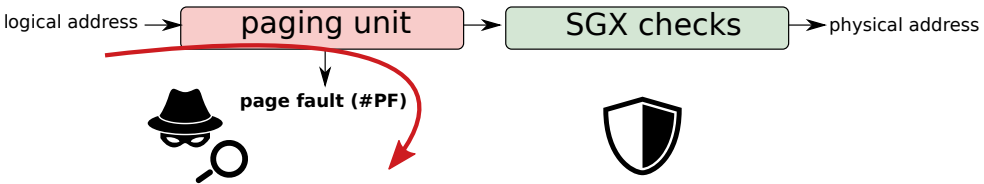
Costan et al. "Intel SGX explained", IACR 2016.

Intel SGX: Page faults as a side channel



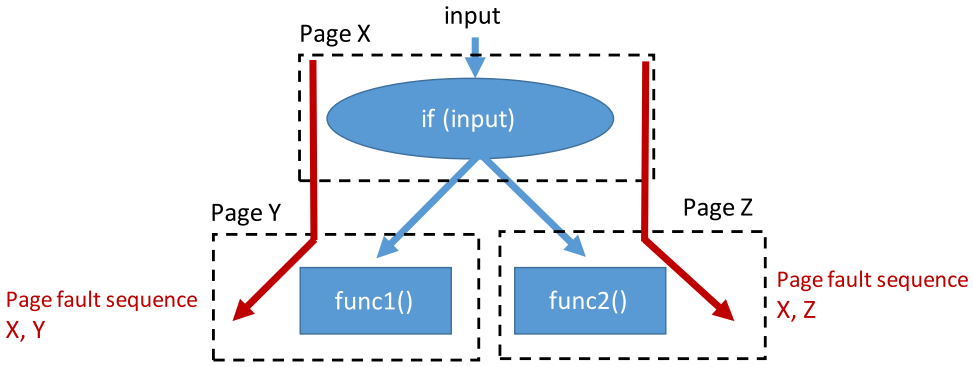
SGX machinery protects against direct address remapping attacks

Intel SGX: Page faults as a side channel



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

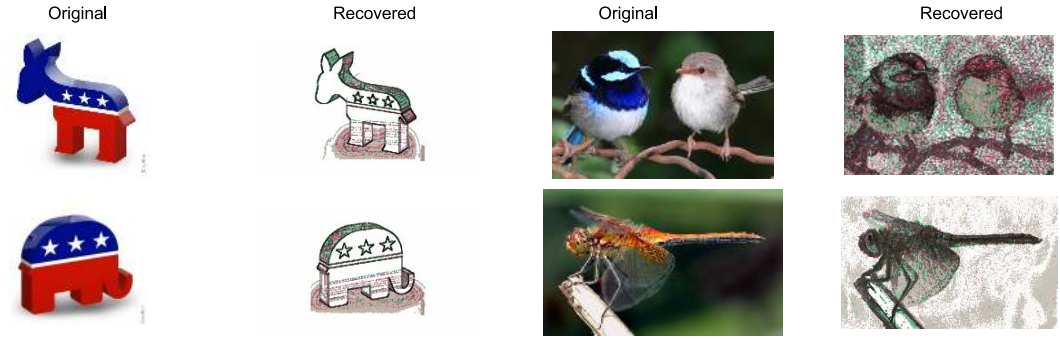
Intel SGX: Page faults as a side channel



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

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

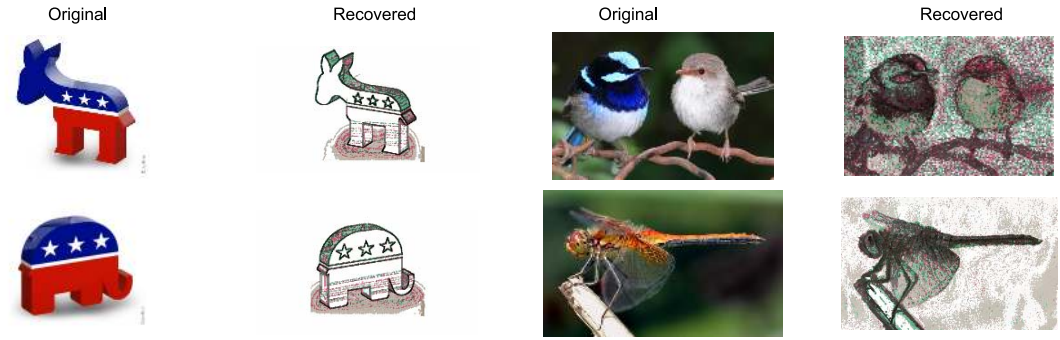
Page table-based attacks in practice



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



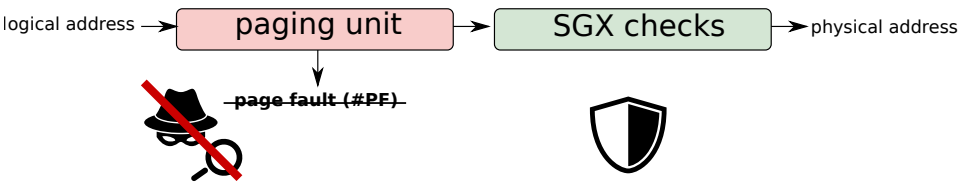
Xu et al.: "Controlled-channel attacks: Deterministic side channels for untrusted operating systems", Oakland 2015.

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

1. Privileged Side-Channel Attacks

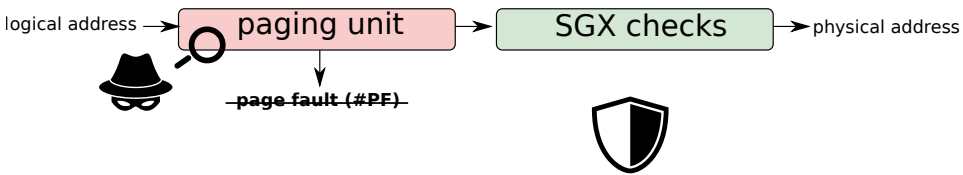
Idea #2: Monitoring without Page Faults?

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.

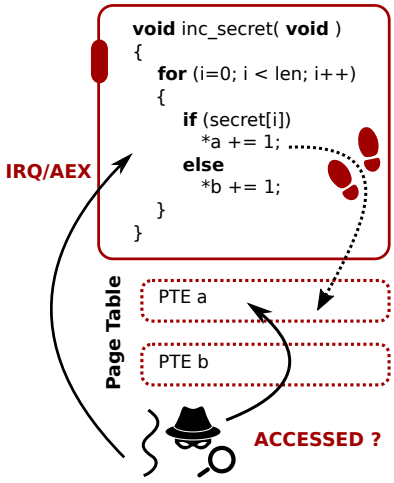
Whenever the processor uses a paging-structure entry as part of linear-address translation, it sets the accessed flag in that entry (if it is not already set).

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).

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.

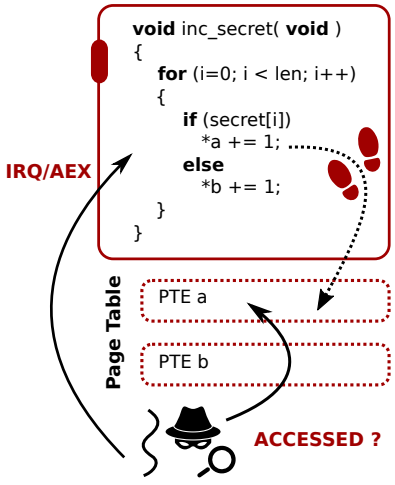
Telling your secrets without page faults

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!

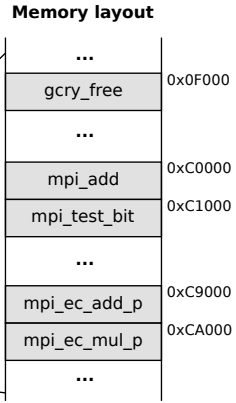


Van Bulck et al. "Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution", USENIX 2017.

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 }
```

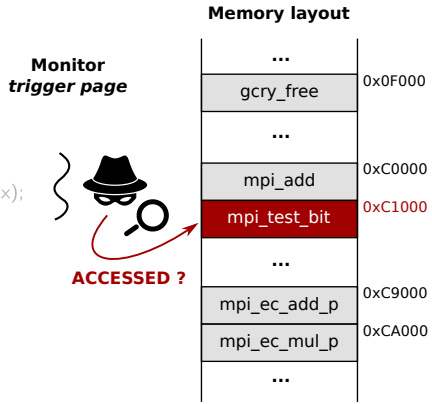
22 Code pages per iteration



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 }
```



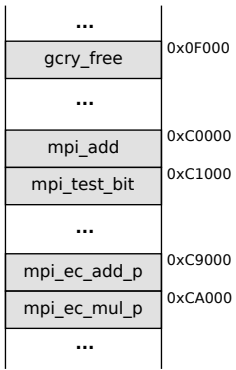
Van Bulck et al. "Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution", USENIX 2017.

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

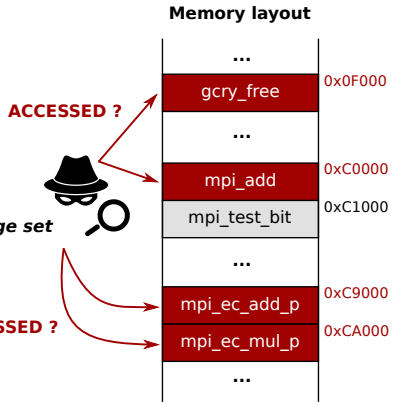
Memory layout



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.

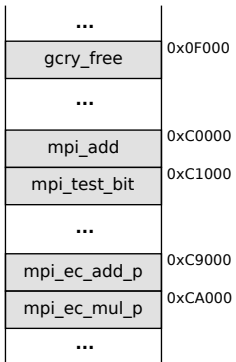
Attacking Libgcrypt EdDSA (simplified)

```
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

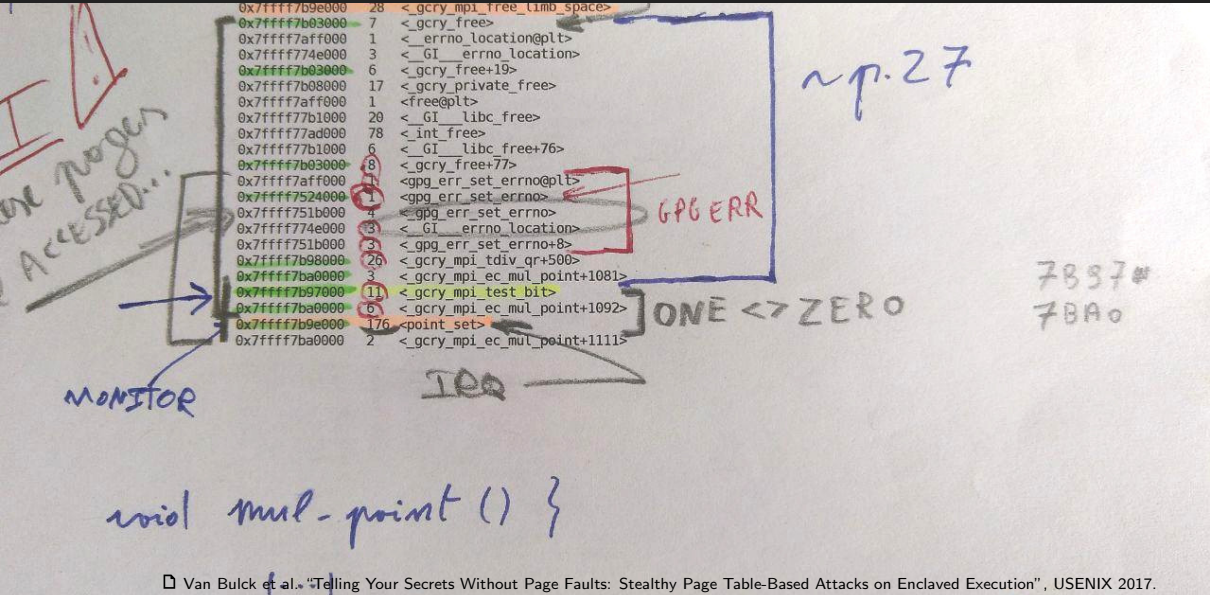
Full 512-bit key recovery, single run

Memory layout



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

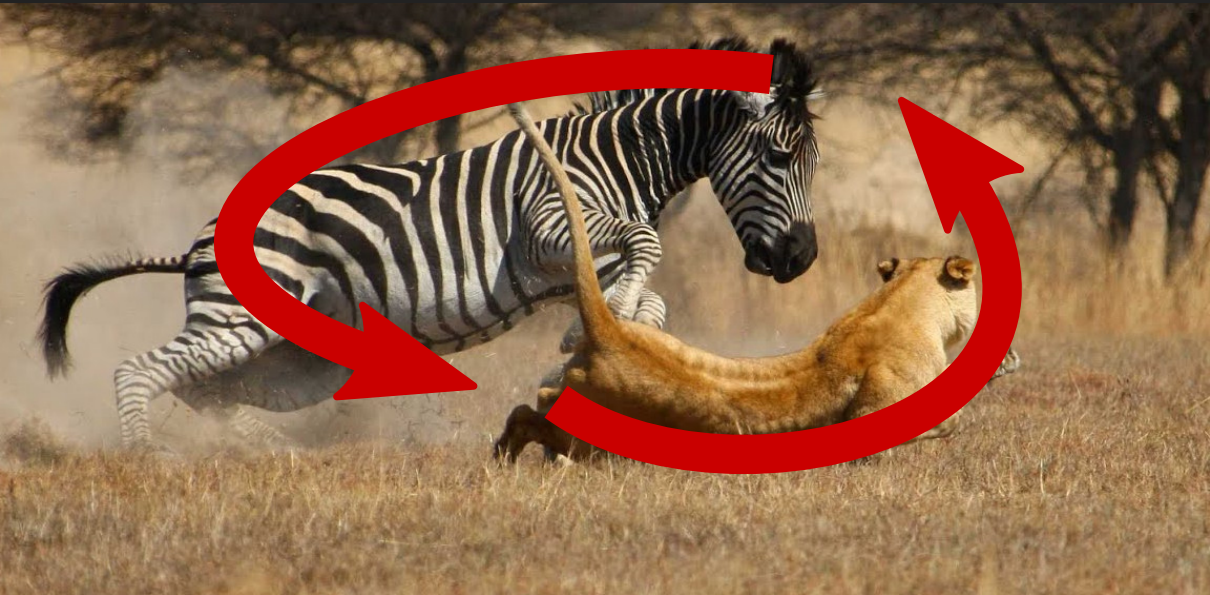
Side-channel analysis: From metadata patterns to secrets



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 . . .

Scientific Understanding Driven by Attacker-Defender Race . . .



1. Privileged Side-Channel Attacks

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.

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);
}
```

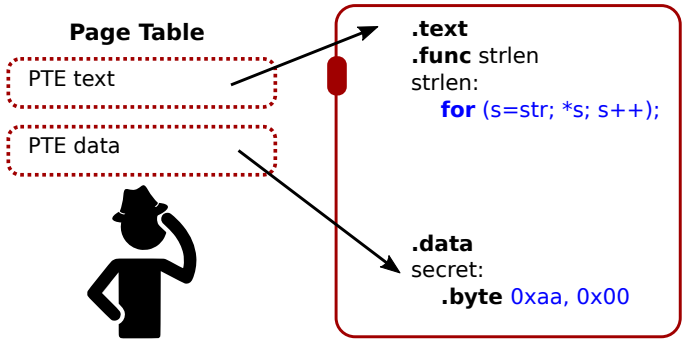
```
    mov  %rdi,%rax
1:  cmpb $0x0,(%rax)
    je   2f
    inc  %rax
    jmp  1b
2:  sub  %rdi,%rax
    retq
```

⇒ 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

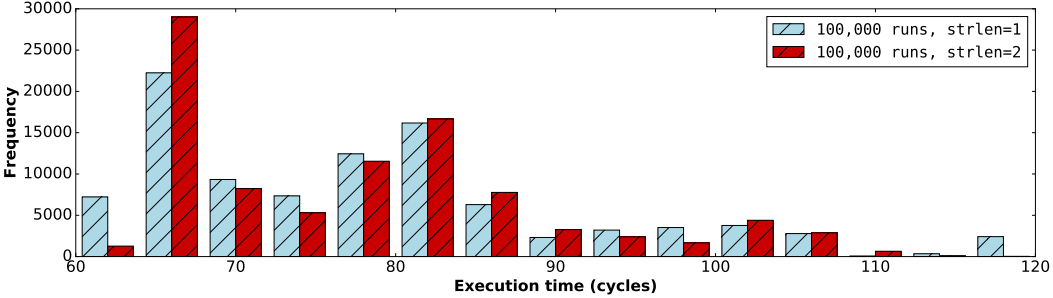
Temporal resolution limitations for the page fault oracle



Counting `strlen` loop iterations?

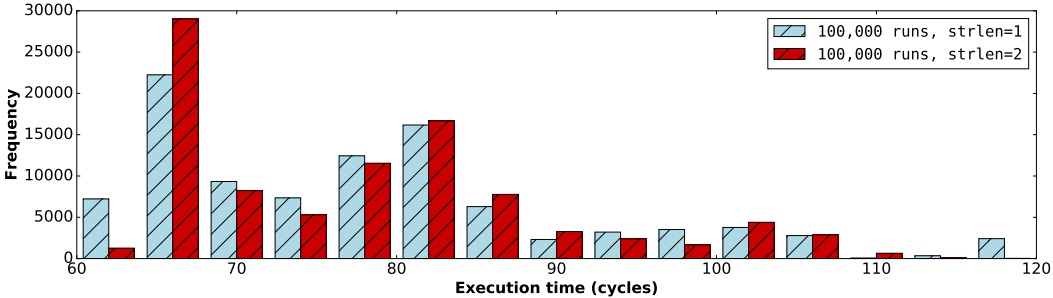
Progress requires both pages present (non-faulting) ↔ page fault oracle

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...



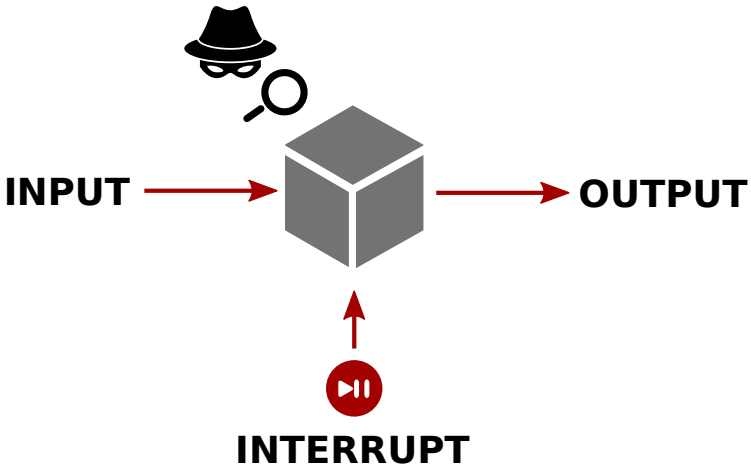
Challenge: Side-channel Sampling Rate

Slow
shutter speed

Medium
shutter speed

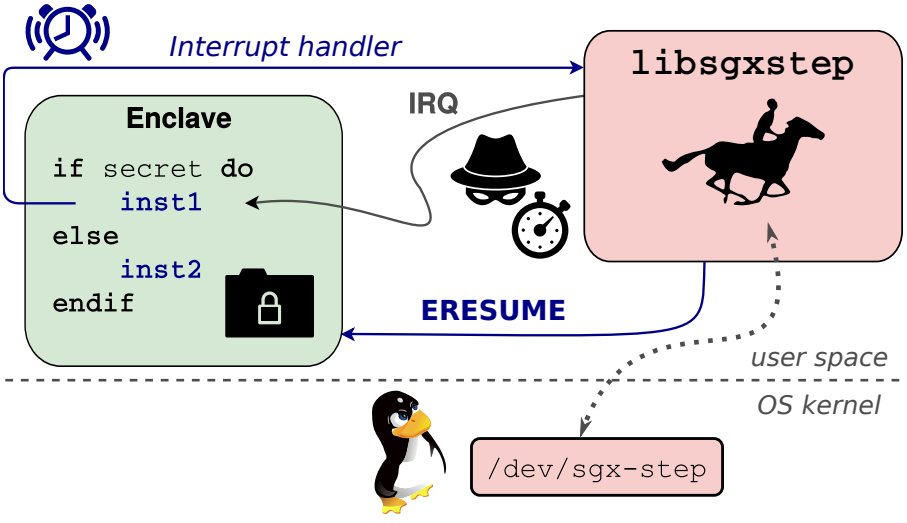
Fast
shutter speed

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 demo: Building a `memcmp()` Password Oracle

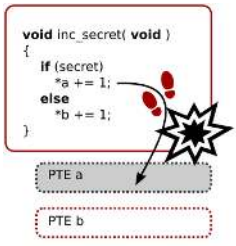
```
[idt.c] DTR.base=0xfffffe0000000000/size=4095 (256 entries)
[idt.c] established user space IDT mapping at 0x7f7ff8e9a000
[idt.c] installed asm IRQ handler at 10:0x56312d19b000
[idt.c] IDT[ 45] @0x7f7ff8e9a2d0 = 0x56312d19b000 (seg sel 0x10); p=1; dpl=3; type=14; ist=0
[file.c] reading buffer from '/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$
```

SGX-Step: Enabling a New Line of High-Resolution Attacks

Yr	Venue	Paper	Step	Use Case	Drv
'15	S&P	Ctrl channel	~ Page	*Probe* (page fault)	✓
'16	ESORICS	AsyncShock	~ Page	*Exploit* (mem safety)	–
'17	CHES	CacheZoom	✗ >1	*Probe* (L1 cache)	✓
'17	ATC	Hahnel et al.	✗ 0 - >1	*Probe* (L1 cache)	✓
'17	USENIX	BranchShadow	✗ 5 - 50	*Probe* (BPU)	✗
'17	USENIX	Stealthy PTE	~ Page	*Probe* (page table)	✓
'17	USENIX	DarkROP	~ Page	*Exploit* (mem safety)	✓
'17	SysTEX	SGX-Step	✓ 0 - 1	*Framework*	✓
'18	ESSoS	Off-limits	✓ 0 - 1	*Probe* (segmentation)	✓
'18	AsiaCCS	Single-trace RSA	~ Page	*Probe* (page fault)	✓
'18	USENIX	Foreshadow	✓ 0 - 1	*Probe* (transient exec)	✓
'18	EuroS&P	SgxPectre	~ Page	*Exploit* (transient)	✓
'18	CHES	CacheQuote	✗ >1	*Probe* (L1 cache)	✓
'18	ICCD	SGXlinger	✗ >1	*Probe* (IRQ latency)	✗
'18	CCS	Nemesis	✓ 1	*Probe* (IRQ latency)	✓
'19	USENIX	Spoiler	✓ 1	*Probe* (IRQ latency)	✓
'19	CCS	ZombieLoad	✓ 0 - 1	*Probe* (transient exec)	✓
'19	CCS	Fallout	–	*Probe* (transient exec)	✓
'19	CCS	Tale of 2 worlds	✓ 1	*Exploit* (mem safety)	✓
'19	ISCA	MicroScope	~ 0 - Page	*Framework*	✗
'20	CHES	Bluethunder	✓ 1	*Probe* (BPU)	✓
'20	USENIX	Big troubles	~ Page	*Probe* (page fault)	✓
'20	S&P	Plundervolt	–	*Exploit* (undervolt)	✓
'20	CHES	Viral primitive	✓ 1	*Probe* (IRQ count)	✓
'20	USENIX	CopyCat	✓ 1	*Probe* (IRQ count)	✓
'20	S&P	LVI	✓ 1	*Exploit* (transient)	✓

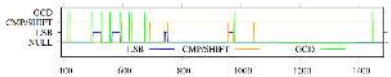
Yr	Venue	Paper	Step	Use Case	Drv
'20	CHES	A to Z	~ Page	*Probe* (page fault)	✓
'20	CCS	Déjà Vu NSS	~ Page	*Probe* (page fault)	✓
'20	MICRO	PTHammer	–	*Probe* (page walk)	✓
'21	USENIX	Frontal	✓ 1	*Probe* (IRQ latency)	✓
'21	S&P	CrossTalk	✓ 1	*Probe* (transient exec)	✓
'21	CHES	Online template	✓ 1	*Probe* (IRQ count)	✓
'21	NDSS	SpeechMiner	–	*Framework*	✓
'21	S&P	Platypus	✓ 0 - 1	*Probe* (voltage)	✓
'21	DIMVA	Aion	✓ 1	*Probe* (cache)	✓
'21	CCS	SmashEx	✓ 1	*Exploit* (mem safety)	✓
'21	CCS	Util::Lookup	✓ 1	*Probe* (L3 cache)	✓
'22	USENIX	Rapid prototyping	✓ 1	*Framework*	✓
'22	CT-RSA	Kalyna expansion	✓ 1	*Probe* (L3 cache)	✓
'22	SEED	Enclyzer	–	*Framework*	✓
'22	NordSec	Self-monitoring	~ Page	*Defense* (detect)	✓
'22	AutoSec	Robotic vehicles	✓ 1 - >1	*Exploit* (timestamp)	✓
'22	ACSAC	MoLE	✓ 1	*Defense* (randomize)	✓
'22	USENIX	AEPIC	✓ 1	*Probe* (I/O device)	✓
'22	arXiv	Confidential code	✓ 1	*Probe* (IRQ latency)	✓
'23	ComSec	FaultMorse	~ Page	*Probe* (page fault)	✓
'23	CHES	HQC timing	✓ 1	*Probe* (L3 cache)	✓
'23	ISCA	Belong to us	✓ 1	*Probe* (BPU)	✓
'23	USENIX	BunnyHop	✓ 1	*Probe* (BPU)	✓
'23	USENIX	DownFall	✓ 0 - 1	*Probe* (transient exec)	✓
'23	USENIX	AEX-Notify	✓ 1	*Defense* (prefetch)	✓

A Versatile Open-Source Attack Toolkit



Interrupt latency

[CCS'18, USENIX'21]



Page-table manipulation

[AsiaCCS'18, USENIX'18-23, CCS20, CHES'20, NDSS'21]

SGX-Step

Interrupt counting

[CCS'19, CHES'20-21, USENIX'20]

High-resolution probing

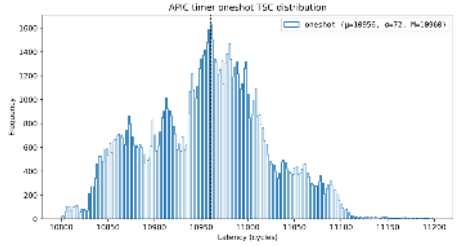
[CCS'19/21, CHES'20, S&P'20-21, USENIX'17/18/22]

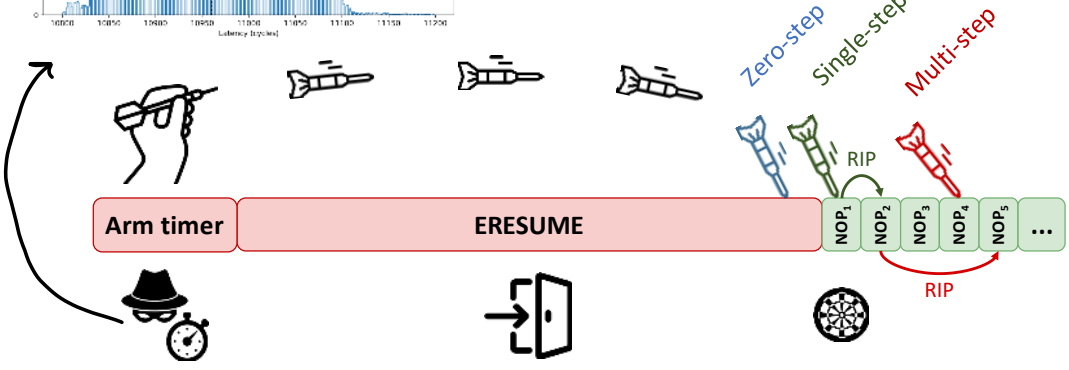
[USENIX'18, CCS'19, S&P'21] Zero-step replaying

1. Privileged Side-Channel Attacks

Idea #4: Interrupt Hardening?

Root-causing SGX-Step: Aiming the timer interrupt





Root-causing SGX-Step: Microcode assists to the rescue!

PTE A-bit	Mean (cycles)	Stddev (cycles)
A=1	27	30
A=0	666	55

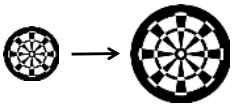
3. Assisted PT walk

1. Clear PTE A-bit

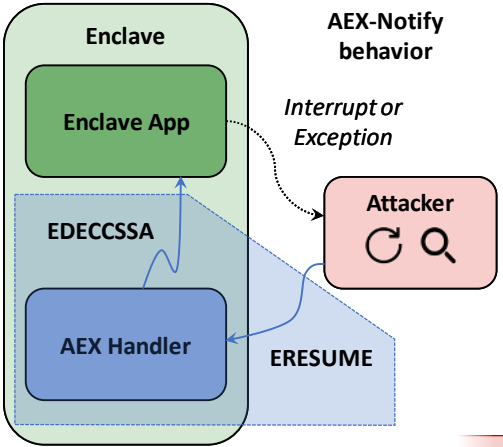
2. TLB flush

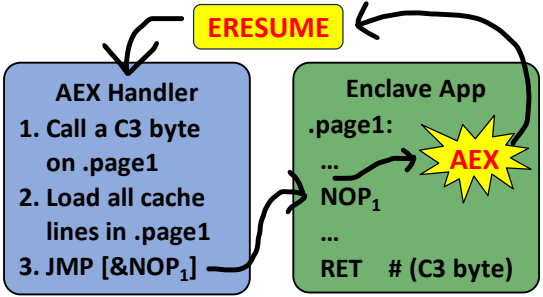
page walk ($RIP) | *exec*

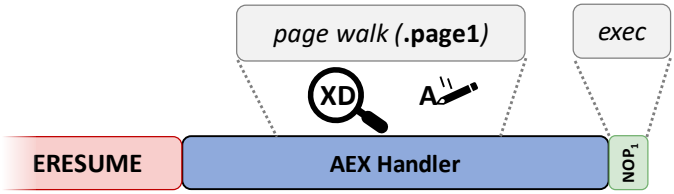
Arm timer | **ERESUME** | **NOP$_1$**



AEX-Notify solution overview

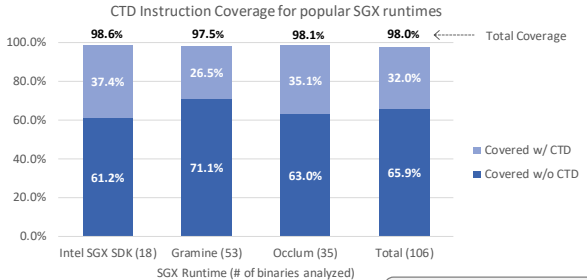


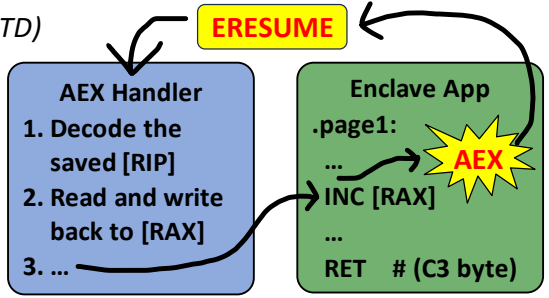


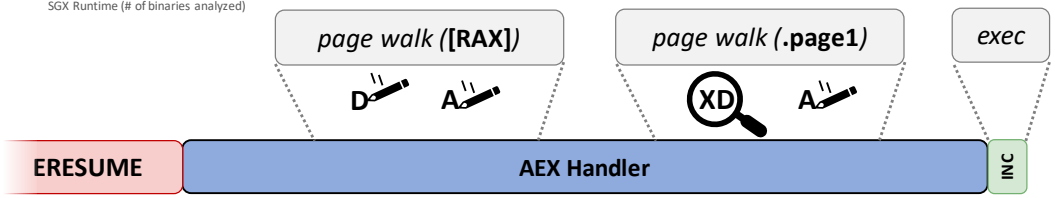


AEX-Notify solution overview

We implemented a fast, constant-time decoder (CTD)







intel

CHAPTER 8
ASYNCHRONOUS ENCLAVE EXIT NOTIFY AND THE EDECCSSA USER LEAF FUNCTION

8.1 INTRODUCTION

Asynchronous Enclave Exit Notify (AEX-Notify) is an extension to Intel® SGX that allows Intel SGX enclaves to be notified after an asynchronous enclave exit (AEX) has occurred. EDECCSSA is a new Intel SGX user leaf function (ENCLU[EDECCSSA]) that can facilitate AEX notification handling, as well as software exception handling. This chapter provides information about changes to the Intel SGX architecture that support AEX-Notify and ENCLU[EDECCSSA].

The following list summarizes the a
details are provided in Section 8.3)

- SECS.ATTRIBUTES.AEXNOTIFY
- TCS.FLAGS.AEXNOTIFY: This e
- SSA.GPRSGX.AEXNOTIFY: Enclave-writable byte that allows enclave software to dynamically enable/disable AEX notifications.

An AEX notification is delivered by ENCLU[ERESUME] when the following conditions are met:

SGX-Step led to **new x86 processor instructions!**

→ shipped in millions of devices ≥ 4th Gen Xeon CPU

phoronix

ARTICLES & REVIEWS | NEWS ARCHIVE | FORUMS | PREMIUM | CONTACT | CATEGORIES

Intel AEX Notify Support Prepped For Linux To Help Enhance SGX Enclave Security

Written by Michael Larabel in Intel on 6 November 2022 at 06:01 AM EST. 5 Comments

Future Intel CPUs and some existing processors via a microcode update will support a new feature called the Asynchronous EXit (AEX) notification mechanism to help with Software Guard Extensions (SGX) enclave security. Patches for the Linux kernel are pending for implementing this Intel AEX Notify support with capable processors.

Intel's Asynchronous EXit (AEX) notification mechanism lets SGX enclaves run a handler after an AEX event. Those handlers can be used for things like mitigating SGX-Step as an attack framework for precise enclave execution control.

Code 1 in intel/linux-sgx Filter

sdk/trts/linux/trts_mitigation.S

```
48   * Description:
49   *      The file provides mitigations for SGX-Step
50   */
71   *  Function:
     constant_time_apply_sgxstep_mitigation_and_continue_execution
72   *      Mitigate SGX-Step and return to the point at which the
     most recent
73   *      interrupt/exception occurred.
```

SGX-Step led to **changes in major OSs and enclave SDKs**

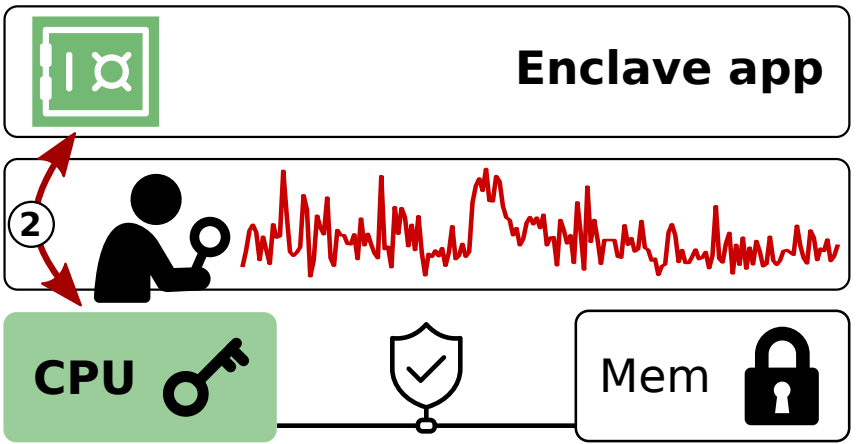
2. Transient-Execution Attacks

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

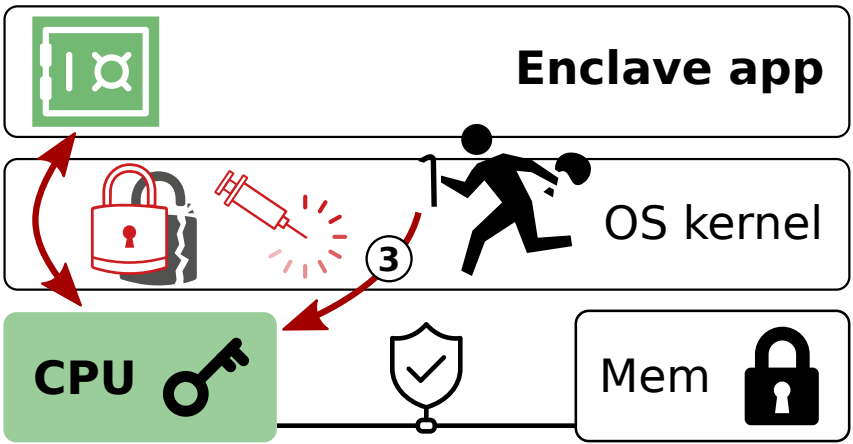
"In general, these research papers do not demonstrate anything new or unexpected about the Intel SGX architecture. Preventing side channel attacks is a matter for the enclave developer. *Intel makes this clear in the security objectives for Intel SGX."*

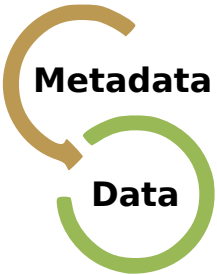
https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sgx-and-side-channels.html

Recap: Privileged side-channel attacks



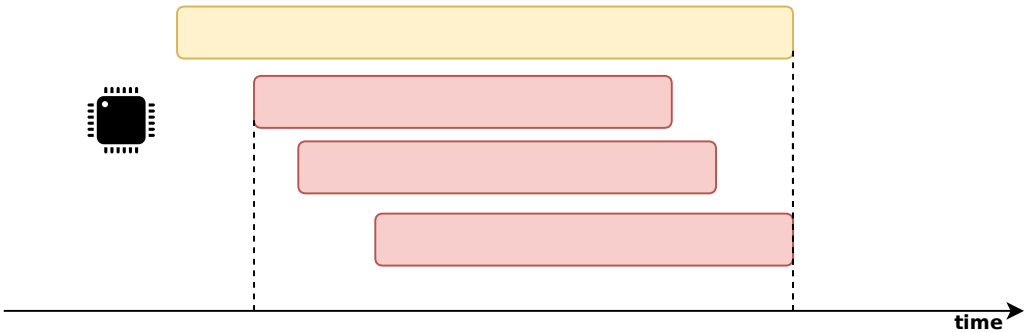
Recap: Transient-execution attacks



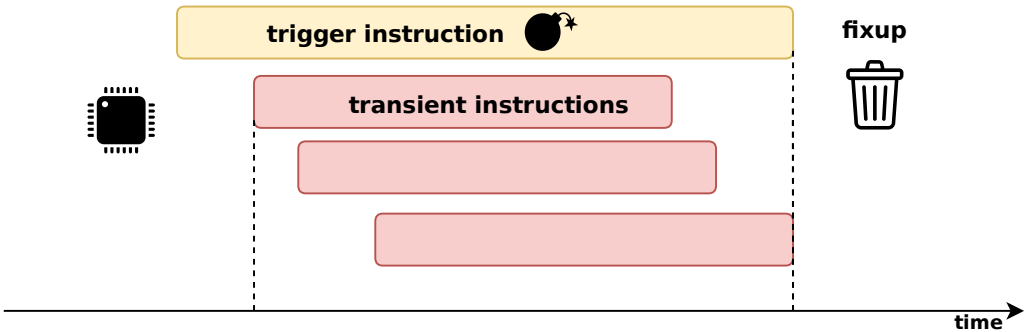



WHAT IF I TOLD YOU
YOU CAN CHANGE RULES MID-GAME
imgflip.com

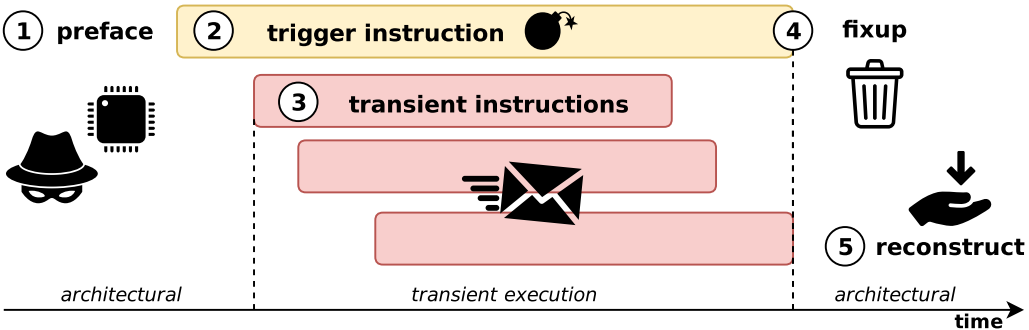
Abusing out-of-order and speculative execution



Abusing out-of-order and speculative execution



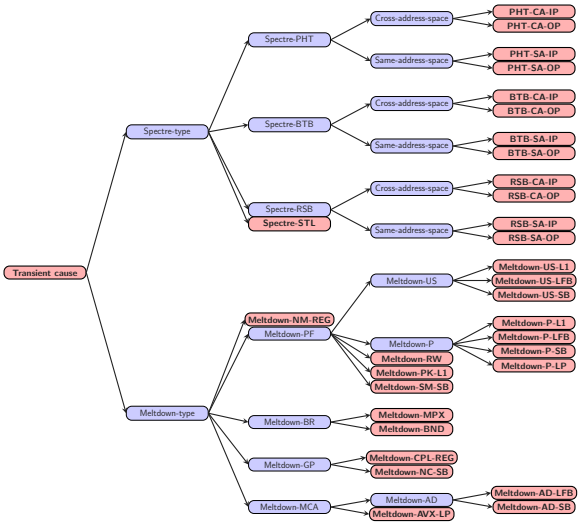
Abusing out-of-order and speculative execution



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

The transient-execution zoo

https://transient.fail



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

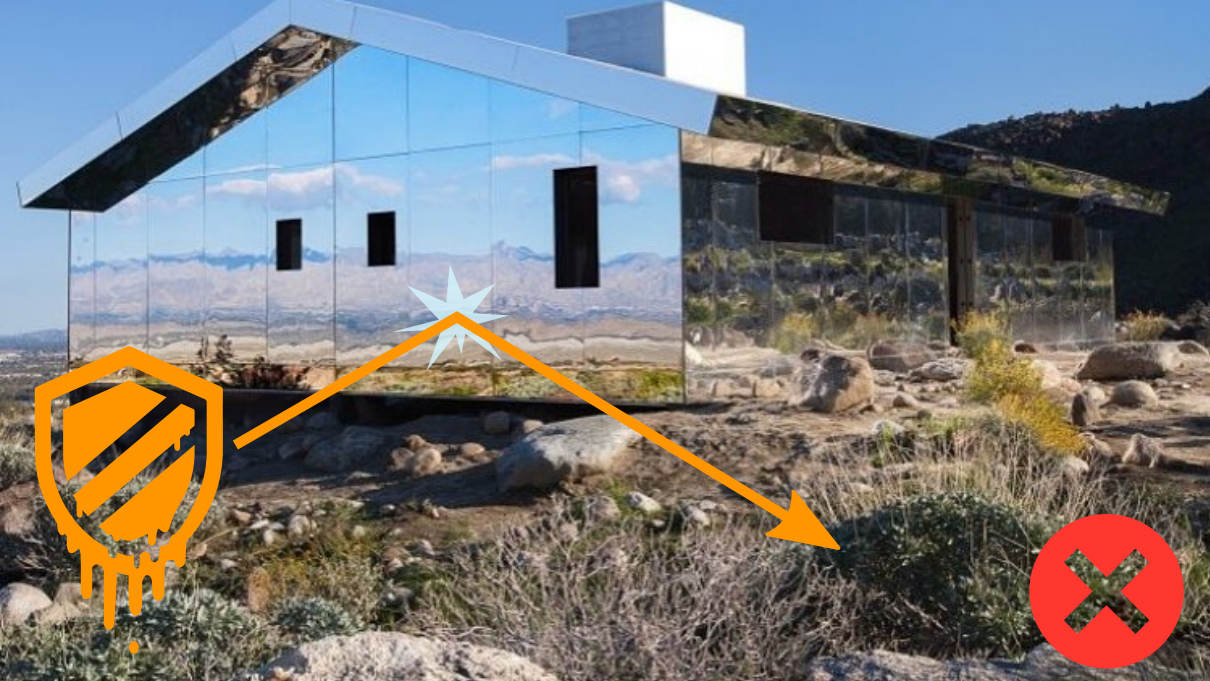
SPECTRE-LIKE FLAW UNDERMINES INTEL PROCESSORS' MOST SECURE ELEMENT

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

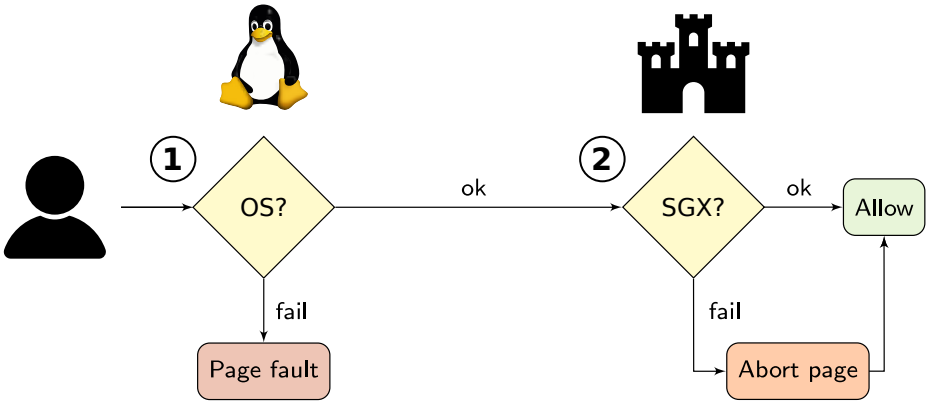
Intel's SGX blown wide open by, you guessed it, a speculative execution attack

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

wired.com and arstechnica.com

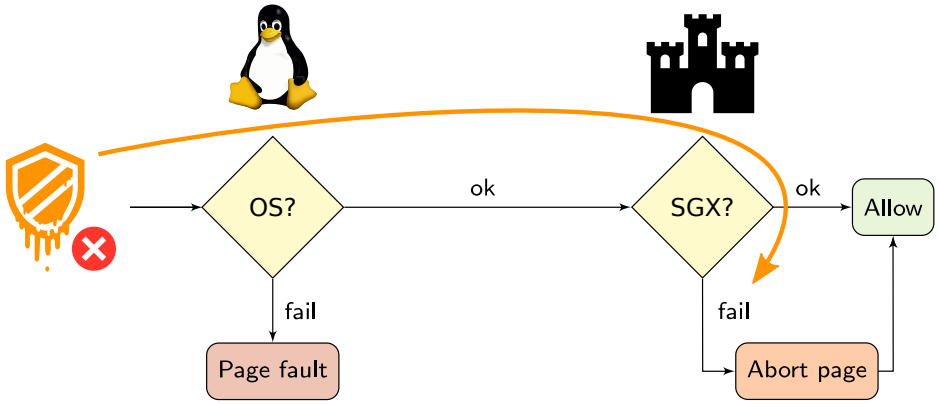


Building Foreshadow: Evade SGX abort page semantics



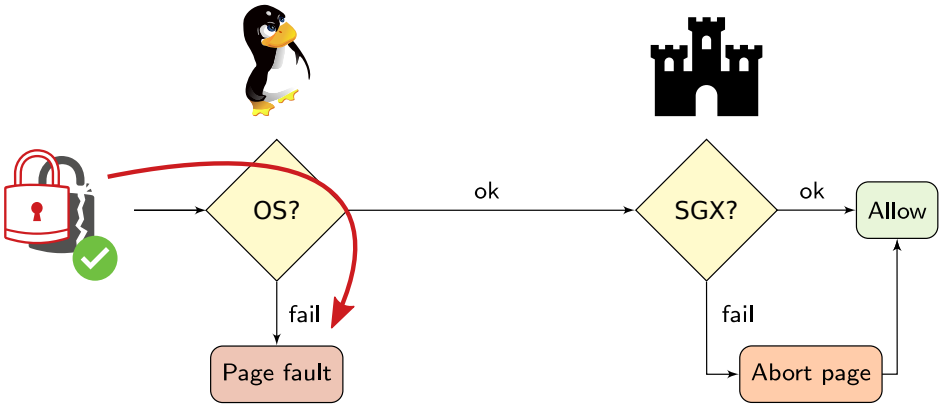
SGX checks prohibit unauthorized access

Building Foreshadow: Evade SGX abort page semantics



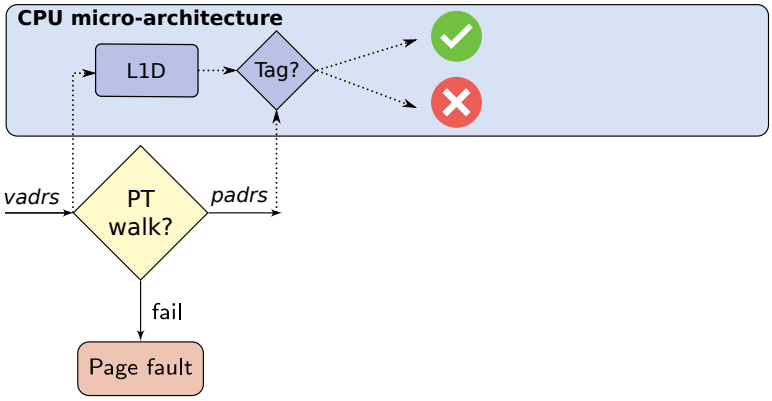
SGX checks prohibit unauthorized access

Building Foreshadow: Evade SGX abort page semantics



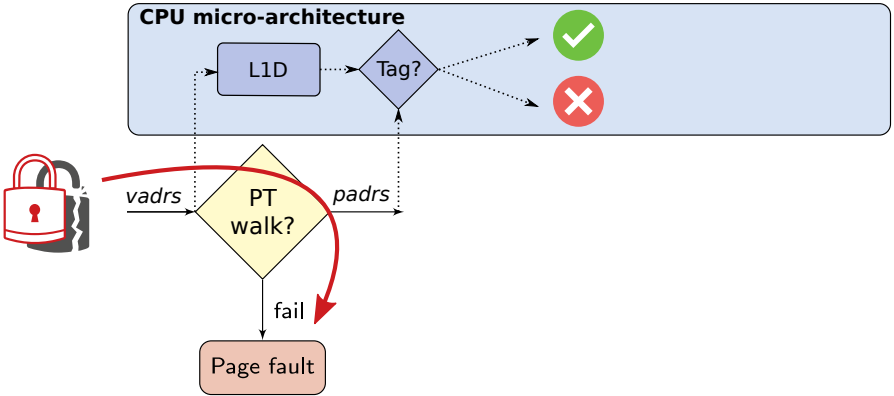
. . . but attackers can **unmap** enclave pages!

The microarchitecture behind Foreshadow



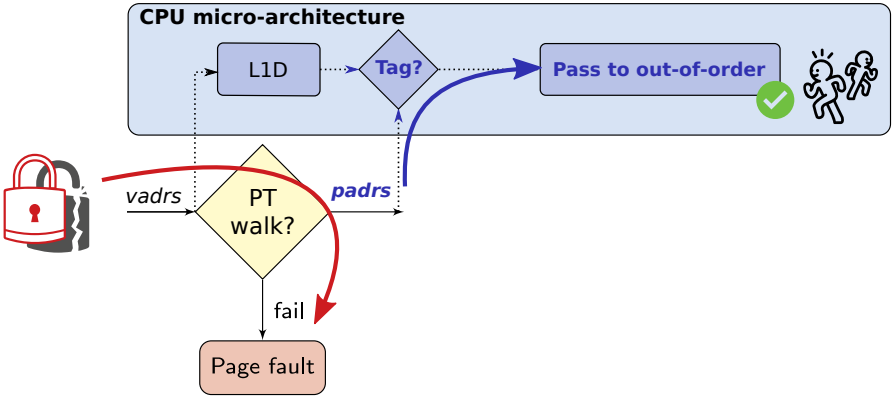
L1 cache design: Virtually-indexed, physically-tagged

The microarchitecture behind Foreshadow



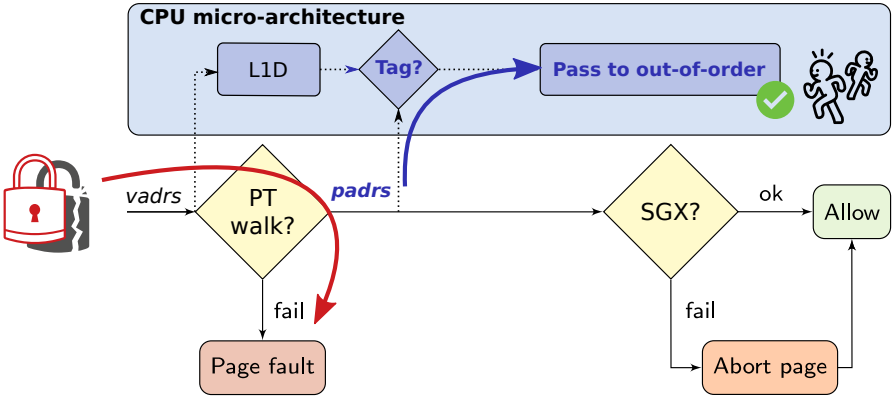
Page fault: Early-out address translation

The microarchitecture behind Foreshadow



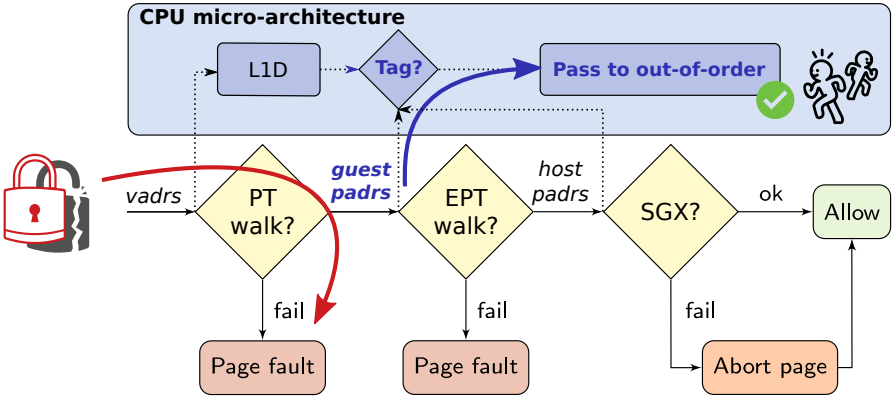
L1-Terminal Fault: Match *unmapped physical address* (!)

The microarchitecture behind Foreshadow

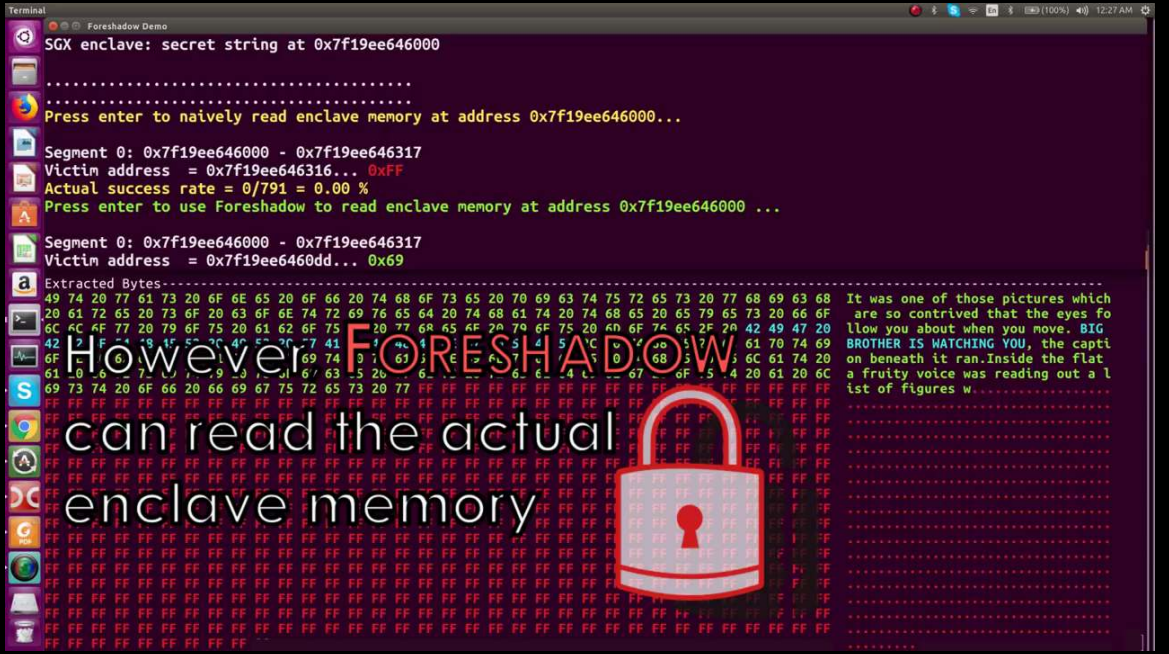


Foreshadow-SGX: Bypass enclave isolation

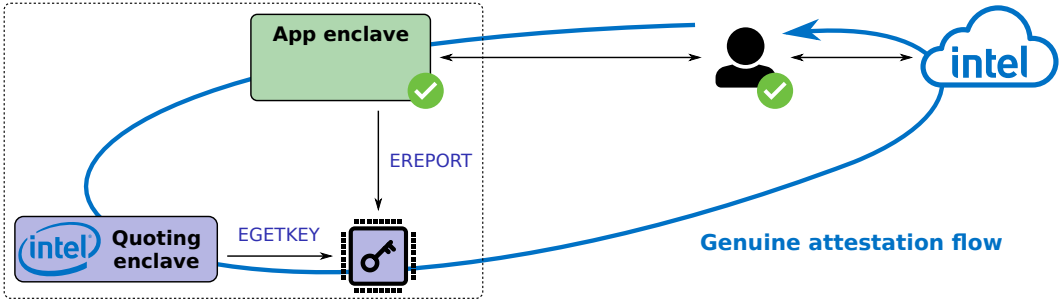
The microarchitecture behind Foreshadow



Foreshadow-NG: Bypass virtual machine isolation

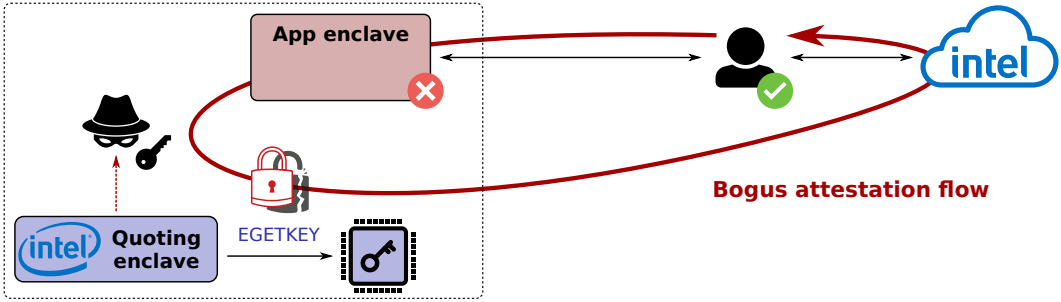

Terminal
Foreshadow Demo
SGX enclave: secret string at 0x7f19ee646000
Press enter to naively read enclave memory at address 0x7f19ee646000...
Segment 0: 0x7f19ee646000 - 0x7f19ee646317
Victim address = 0x7f19ee646316... 0xFF
Actual success rate = 0/791 = 0.00 %
Press enter to use Foreshadow to read enclave 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**)

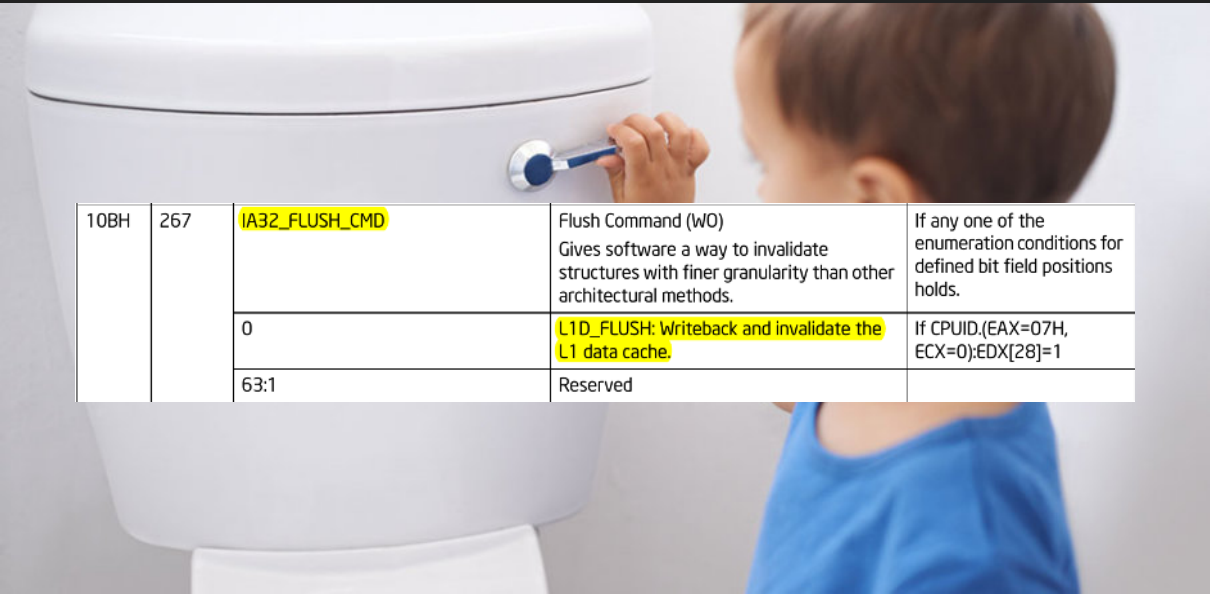
Foreshadow: Extracting the keys to the Intel SGX kingdom



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

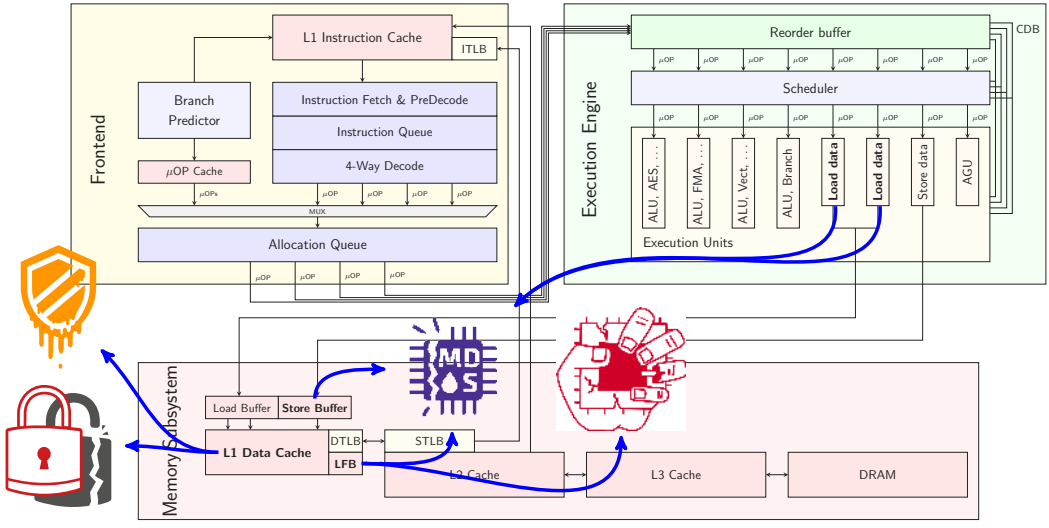
Mitigating Foreshadow: Flush CPU microarchitecture

Mitigating Foreshadow: Flush CPU microarchitecture

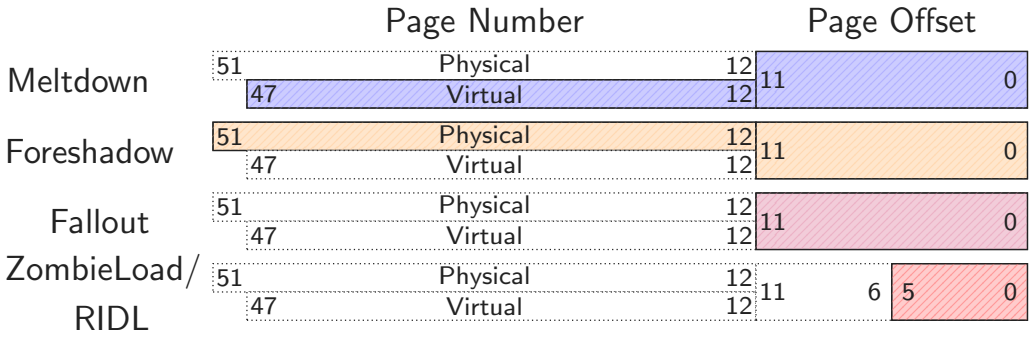


10BH	267	IA32_FLUSH_CMD	Flush Command (WO) Gives software a way to invalidate structures with finer granularity than other architectural methods.	If any one of the enumeration conditions for defined bit field positions holds.
		0	L1D_FLUSH: Writeback and invalidate the L1 data cache.	If CPUID.(EAX=07H,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

MDS variants: Flushing additional microarchitectural buffers



MDS variants: Address dependencies



Take-away: Addressability ≠ (transient) accessibility

Meltdown take-away

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

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

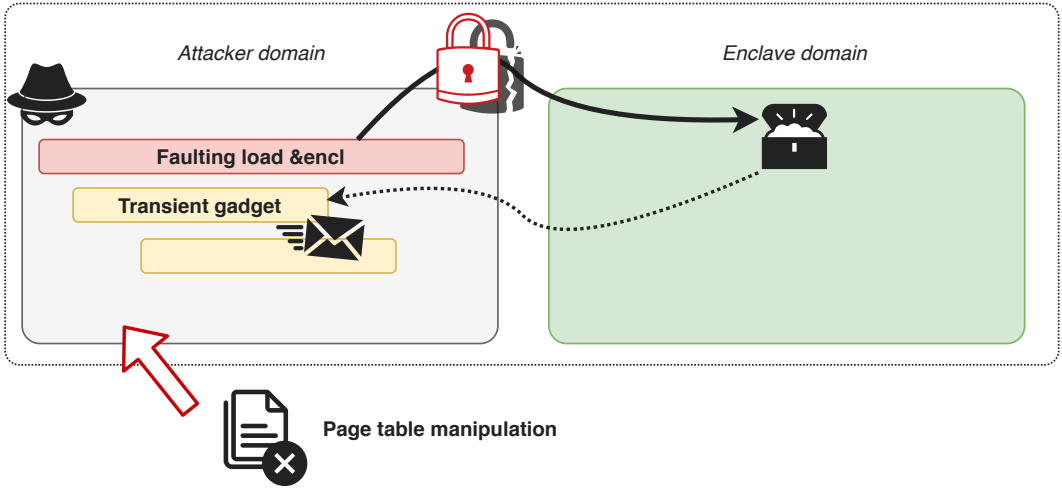
- Meltdown: out-of-reach
- Foreshadow: cache emptied

Intra-enclave view

- Access enclave + outside memory

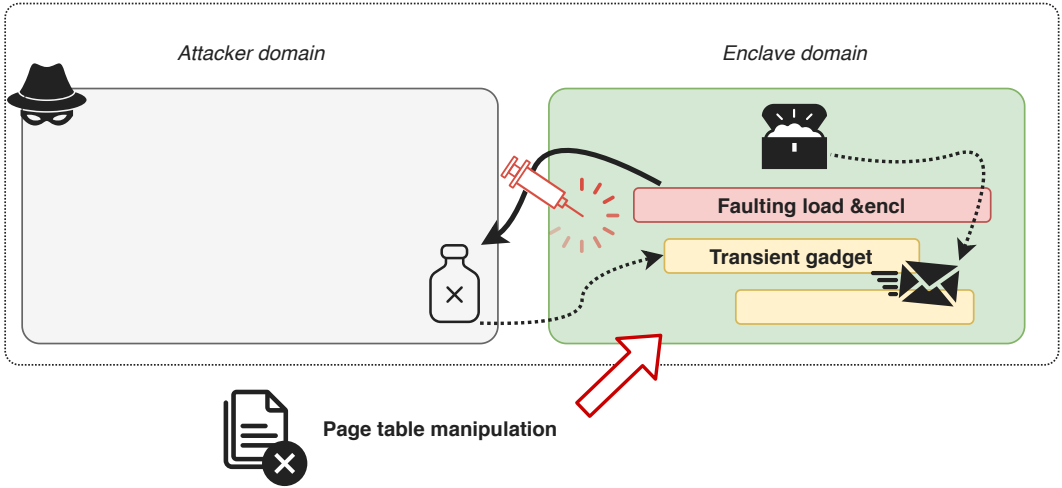
→ Abuse **in-enclave code gadgets!**

Reviving Foreshadow with Load Value Injection (LVI)



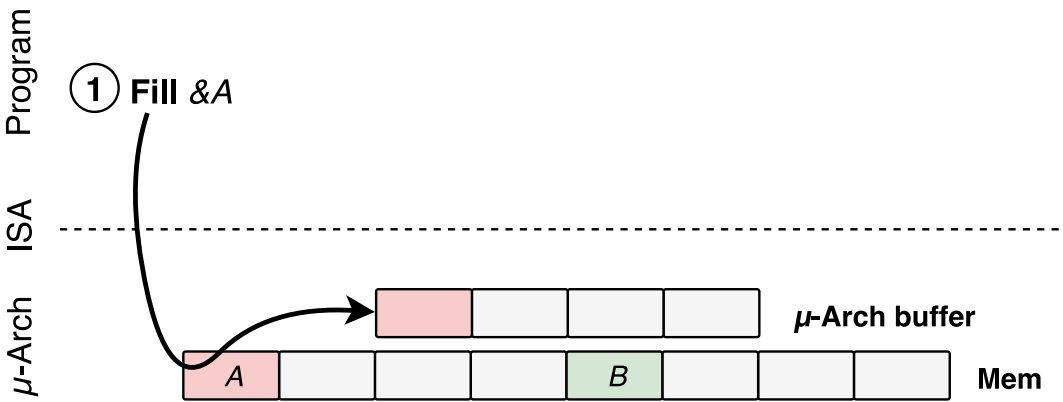
Van Bulck et al. "Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution", USENIX 2018.

Reviving Foreshadow with Load Value Injection (LVI)

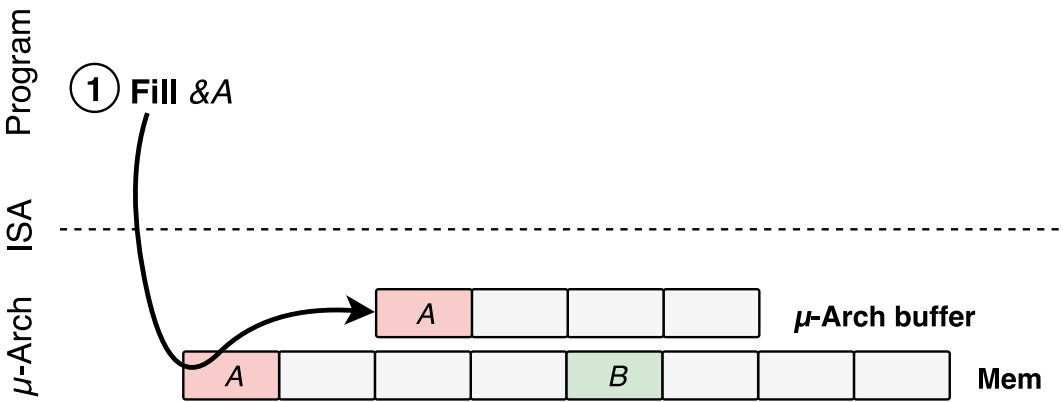


Van Bulck et al. "LVI: Hijacking Transient Execution through Microarchitectural Load Value Injection", S&P 2020.

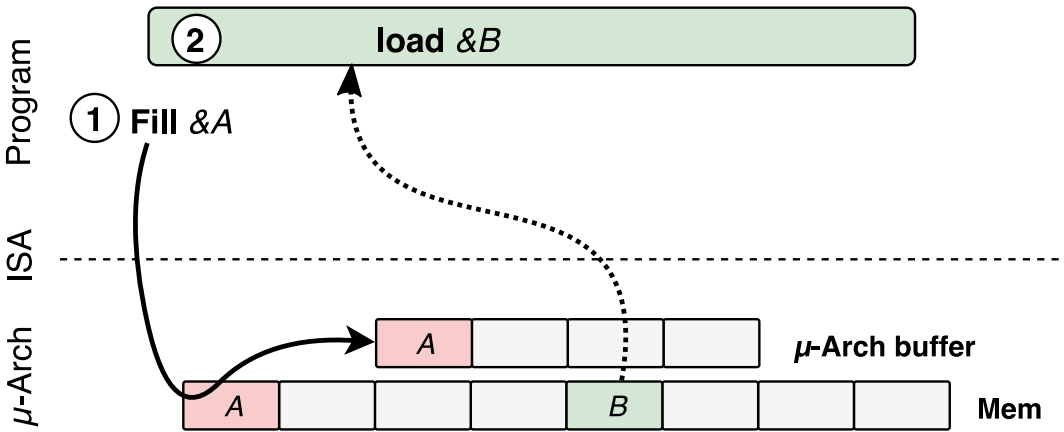
LVI: The basic idea



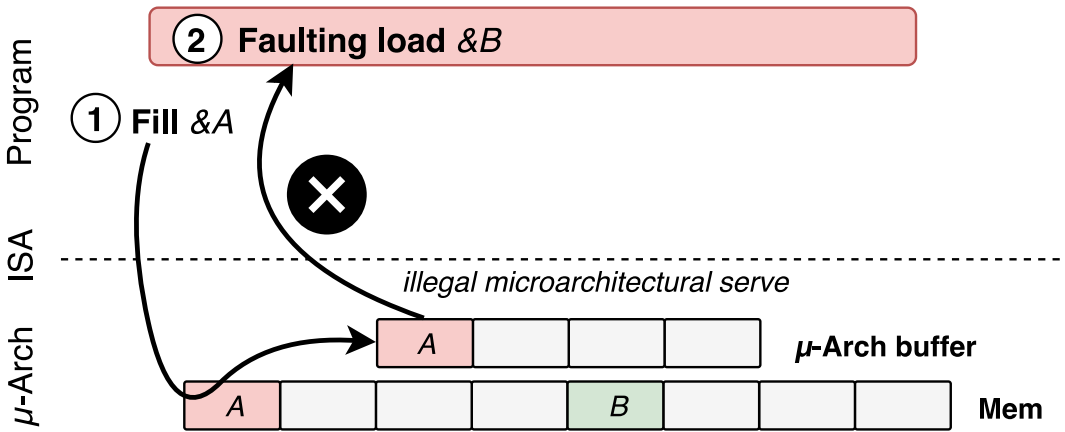
LVI: The basic idea



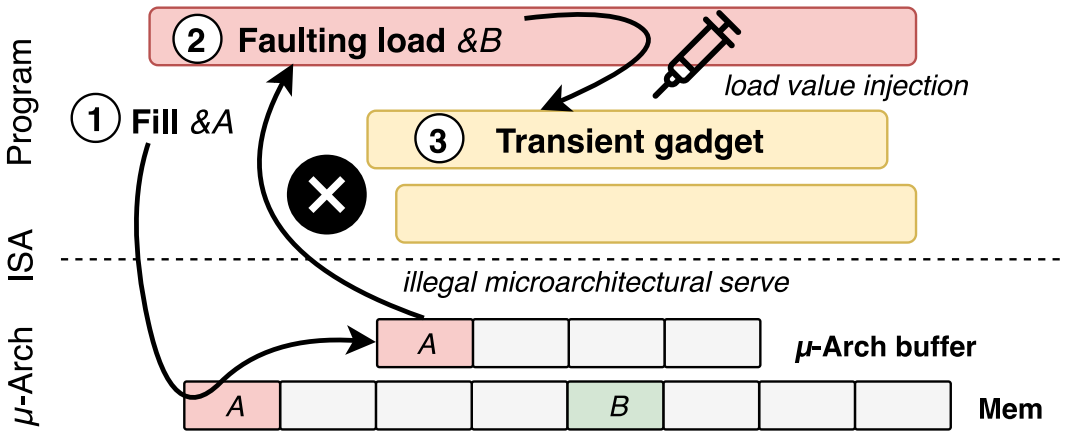
LVI: The basic idea



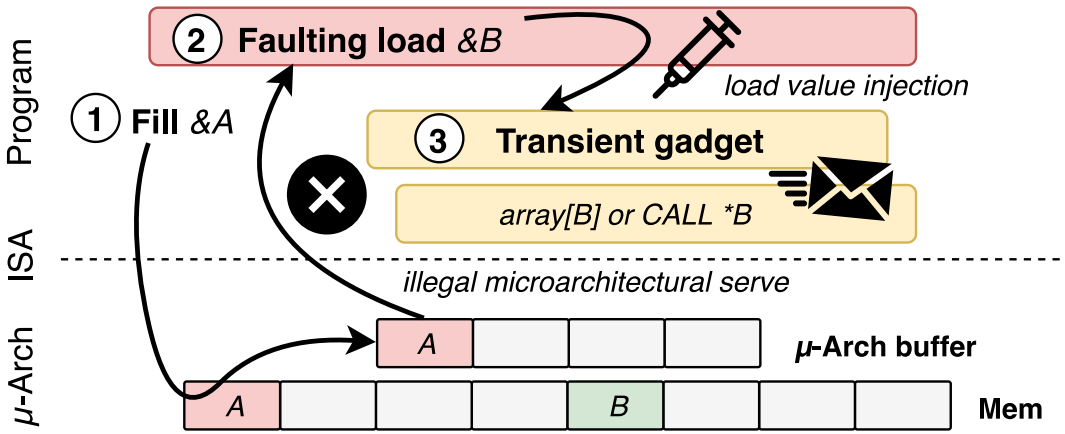
LVI: The basic idea



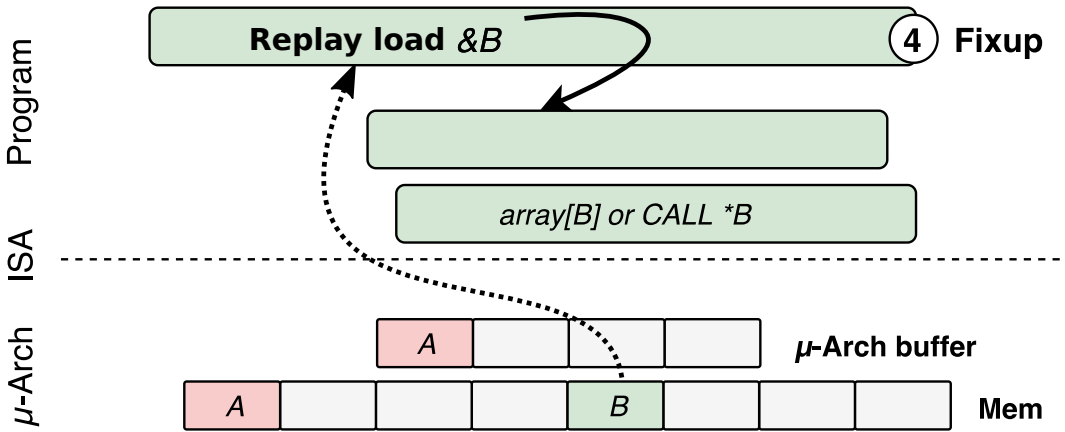
LVI: The basic idea



LVI: The basic idea



LVI: The basic idea

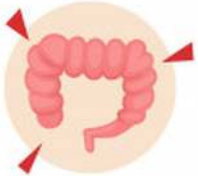


FOOD POISONING

Overdue products

Medicine

Dizziness

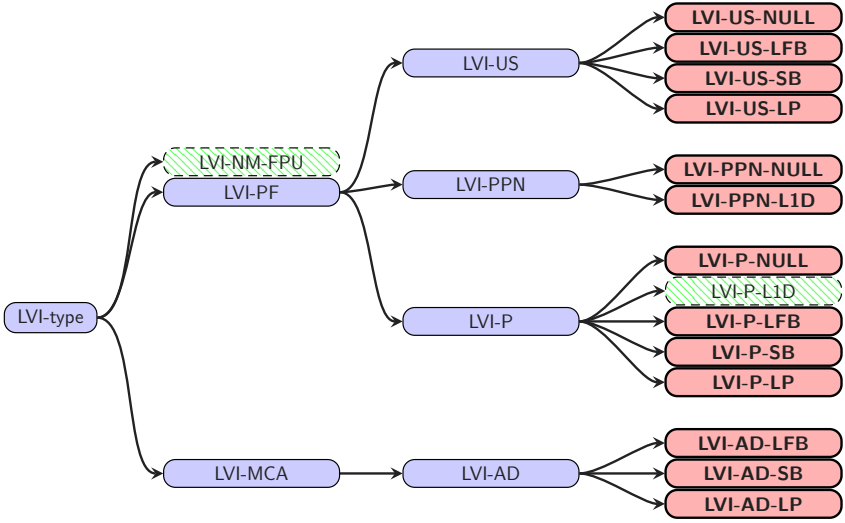


Intestinal colic

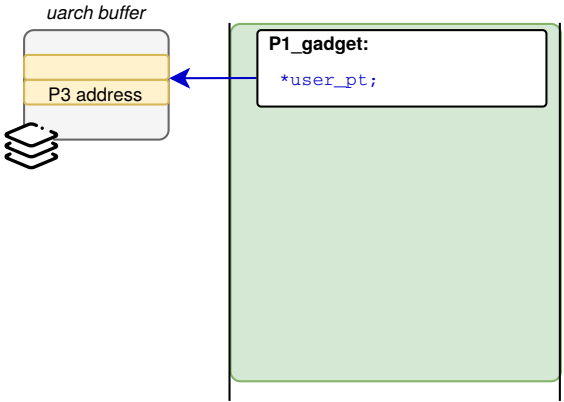
Diarrhea

Headache

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

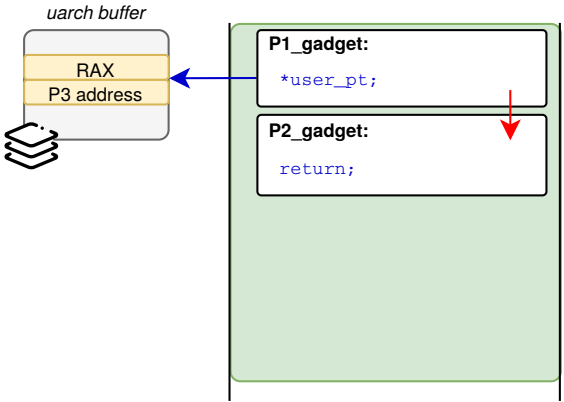


LVI-based transient control-flow hijacking



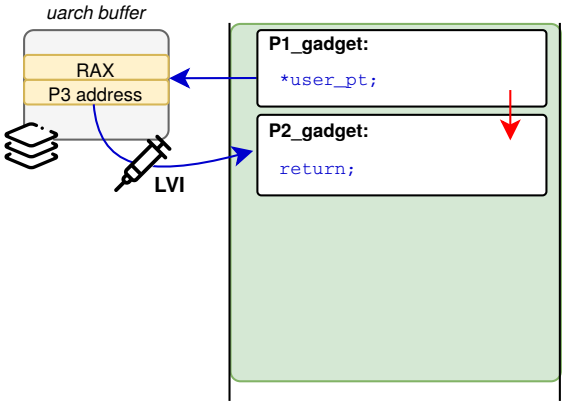
1. Victim fills μ-arch buffer with attacker-controlled data

LVI-based transient control-flow hijacking



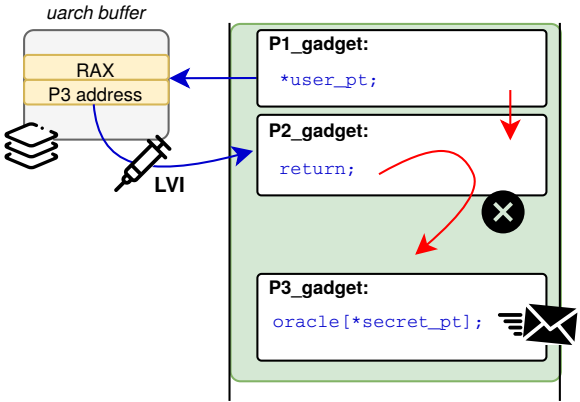
1. Victim fills μ-arch buffer with attacker-controlled data
2. Victim executes indirect branch (`JMP/CALL/RET`)

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
```

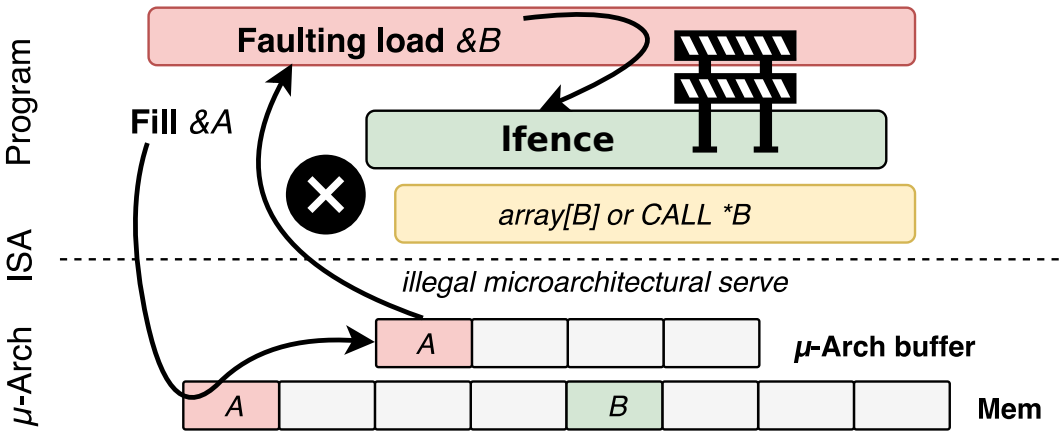
Mitigating LVI: Fencing vulnerable load instructions

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



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

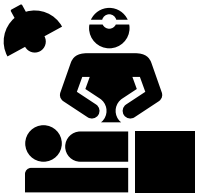
`-Qspectre-load`

More Spectre Mitigations in MSVC

March 13th, 2020

Intel architectural enclaves: `lfence` counts

`libsgx_qe.signed.so`

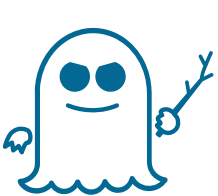


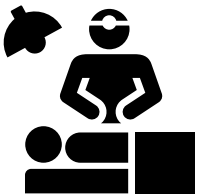
23 fences

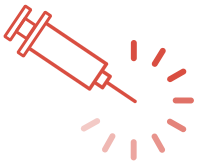
October 2019—"surgical precision"

Intel architectural enclaves: `lfence` counts

`libsgx_qe.signed.so`







23 fences

October 2019—"surgical precision"

49,315 fences

March 2020—"big hammer"

3. Privileged CPU Interface Attacks

Security perimeter for CPU-based isolation?

Security perimeter for CPU-based isolation?

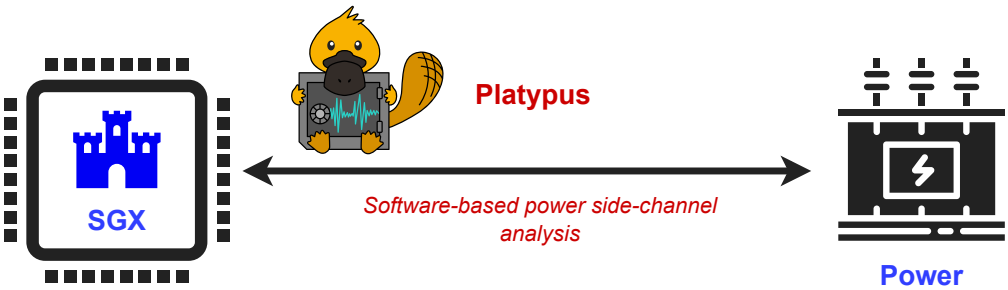
SGX (wants to) only trust the **CPU package**

↔ CPU interfaces with the *physical world(!)*

- (Encrypted) memory bus
- Power supply
- I/O devices

With great power comes great leakage ...

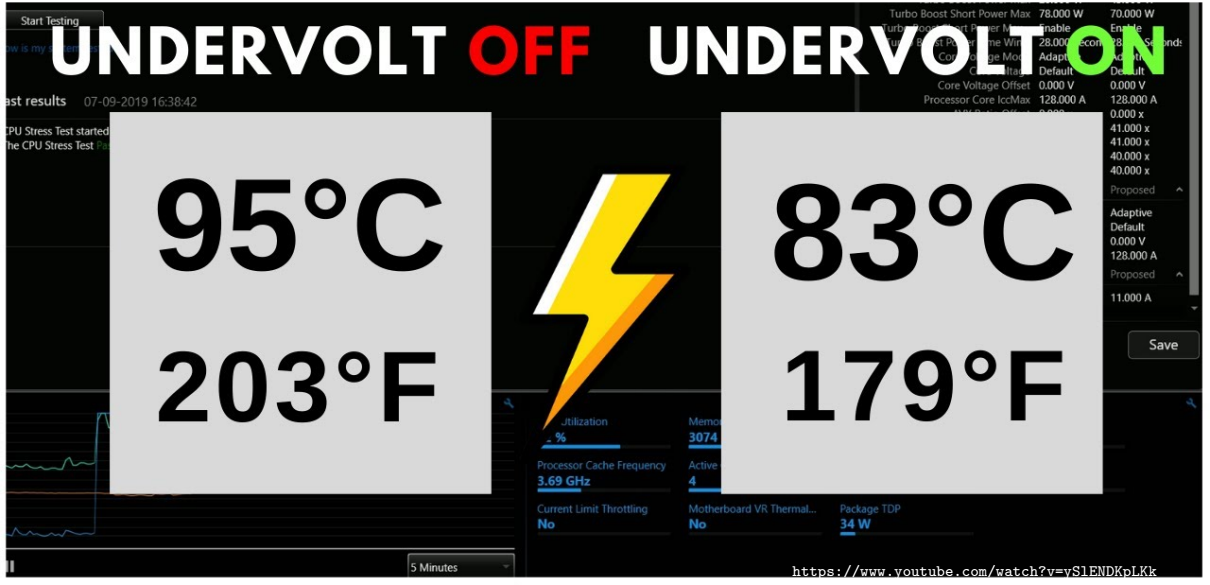
With great power comes great leakage . . .



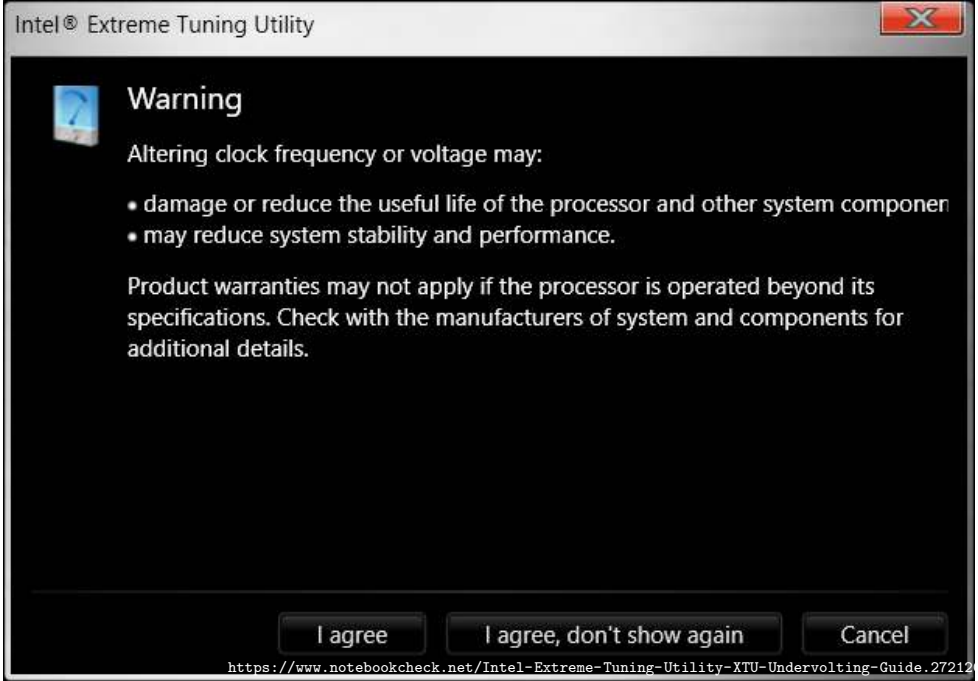
Lipp et al. "PLATYPUS: Software-based Power Side-Channel Attacks on x86", S&P 21.

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

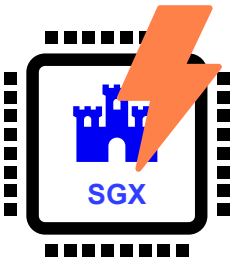
TEMPERATURES I5 9300H



https://www.youtube.com/watch?v=ySlENDKpLKk

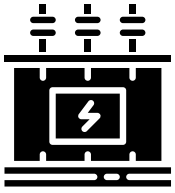

Intel® Extreme Tuning Utility
Warning
Altering clock frequency or voltage may:
• damage or reduce the useful life of the processor and other system componen
• may reduce system stability and performance.
Product warranties may not apply if the processor is operated beyond its specifications. Check with the manufacturers of system and components for additional details.
I agree
I agree, don't show again
Cancel
https://www.notebookcheck.net/Intel-Extreme-Tuning-Utility-XTU-Undervolting-Guide.27212

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

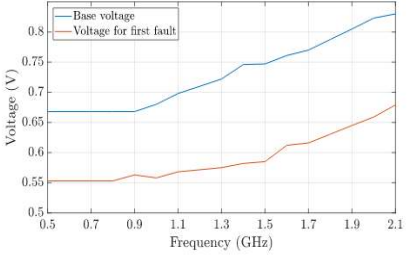


Plundervolt, VoltJockey, V0ltPwn

Software-based fault injection through hidden undervolting MSR

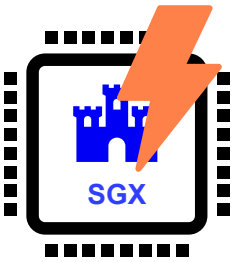


Power

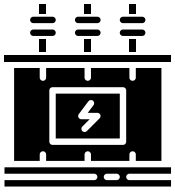


Murdock et al. "Plundervolt: Software-Based Fault Injection Attacks against Intel SGX", S&P 20.

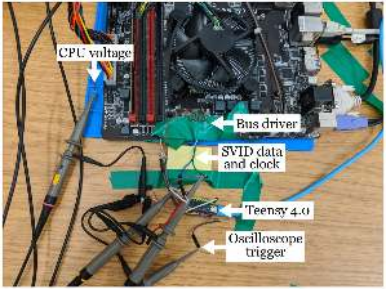
How a little bit of undervolting can cause a lot of problems . . .



Hardware-level physical voltage bus hijacking



Power



< 36$

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

Conclusions and Outlook

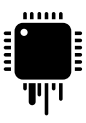
DALL·E 3

Takeaways

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

⇒ Privileged adversaries: Subtle **side channels** can go a long way...

⇒ Scientific understanding driven by **attacker-defender race**

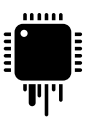


Takeaways

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

⇒ Privileged adversaries: Subtle **side channels** can go a long way...

⇒ Scientific understanding driven by **attacker-defender race**



Thank you! Questions?