~~Secure~~ Trusted Software and Hacking

Jo Van Bulck

Guest Lecture – May 21, 2025

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

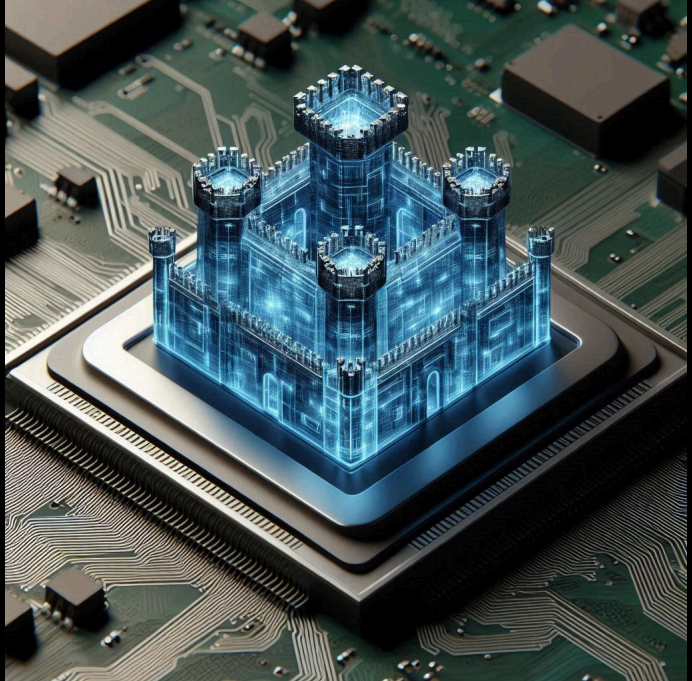
- SSL/TLS etc.

Data in use

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

Data at rest

- Full disk encryption



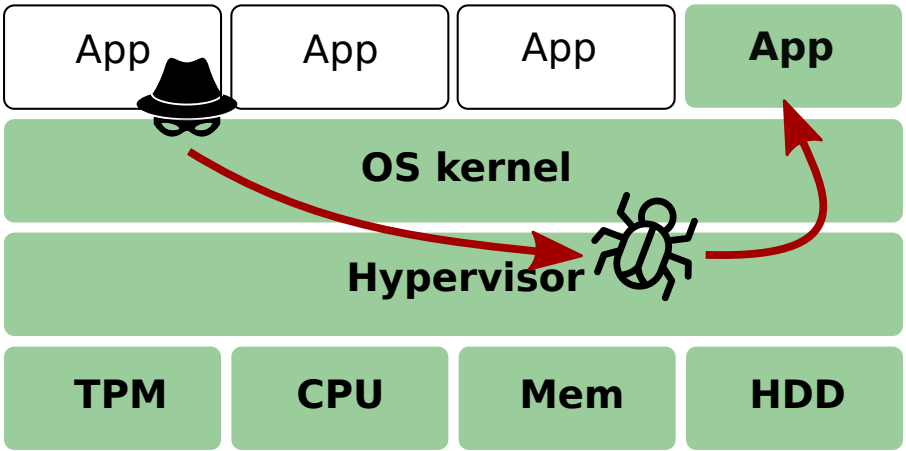
DALL·E 3

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

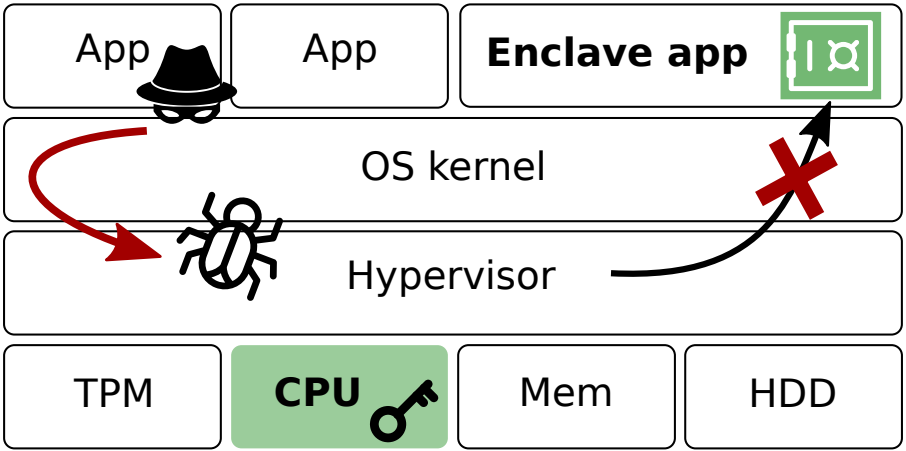
TEEs are here to stay...

Goal: Reducing Attack Surface with Enclaves



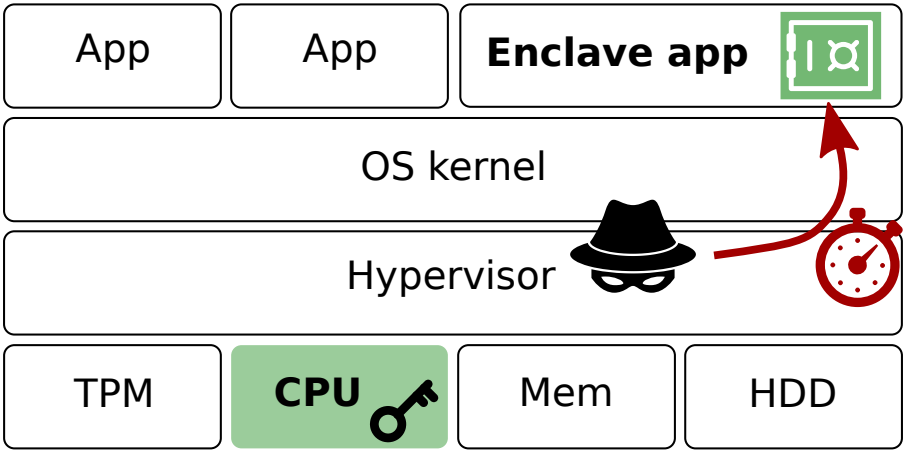
Traditional **layered designs:** Large trusted computing base

Goal: Reducing Attack Surface with Enclaves



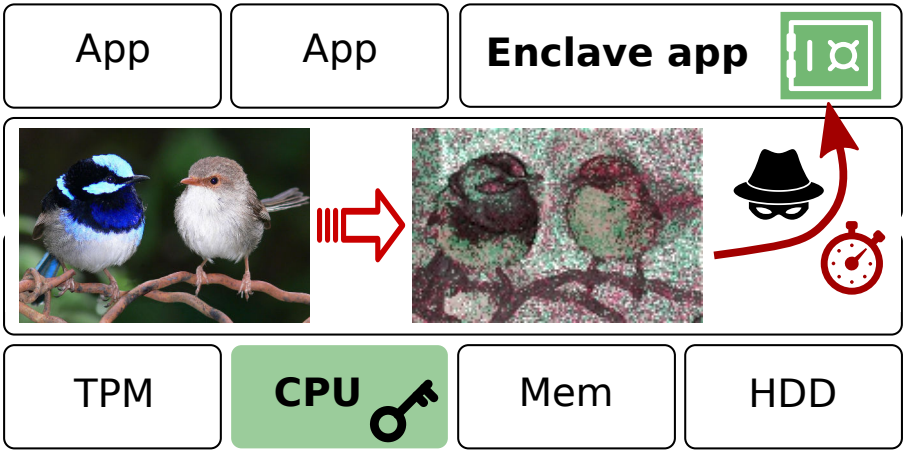
Intel SGX promise: Hardware-level **isolation and attestation**

Reality: Privileged Side-Channel Attacks



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

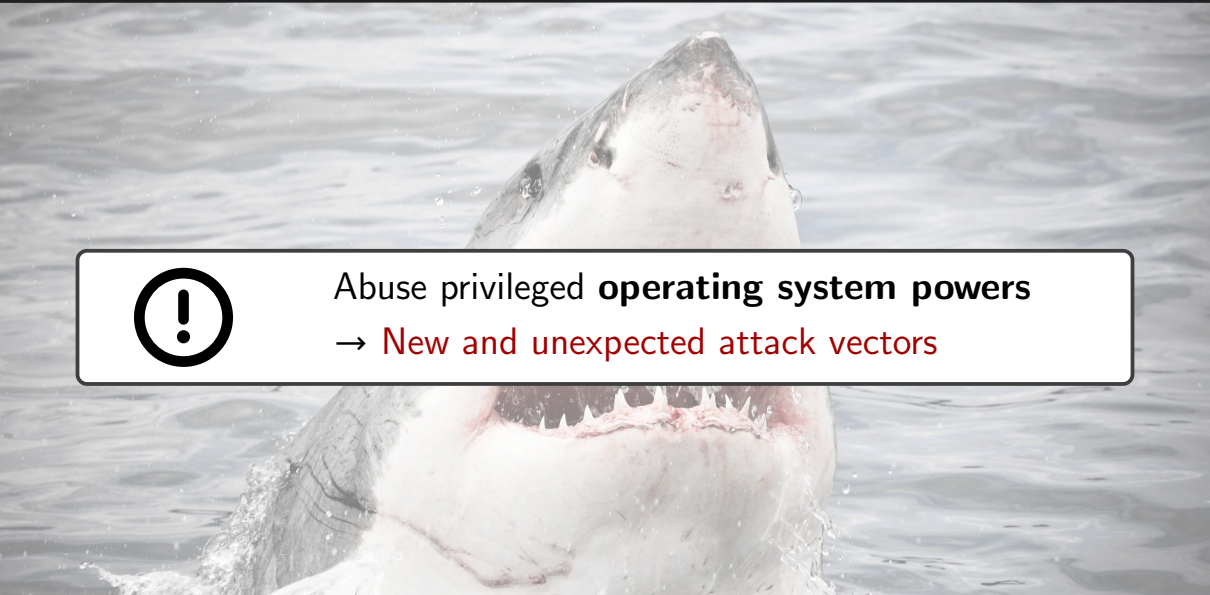
Reality: Privileged Side-Channel Attacks



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

Game Changer: Privileged “Bottom-Up” Adversary Model

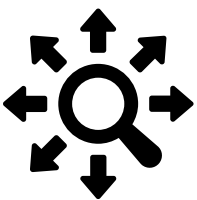
Game Changer: Privileged “Bottom-Up” Adversary Model



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

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 }
```

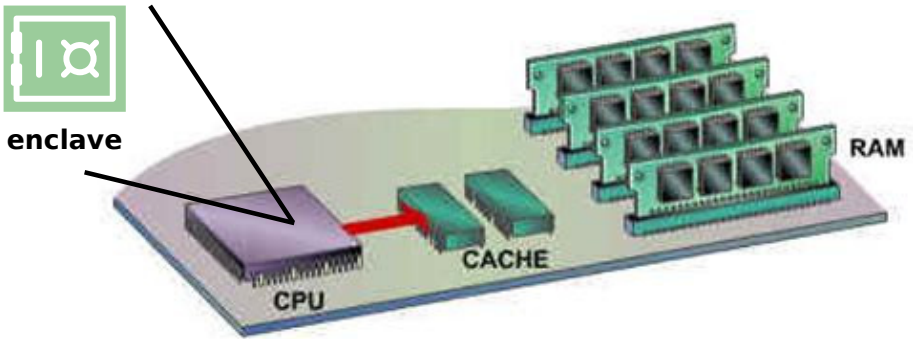
Vulnerable patterns: Secret-dependent code/data accesses

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

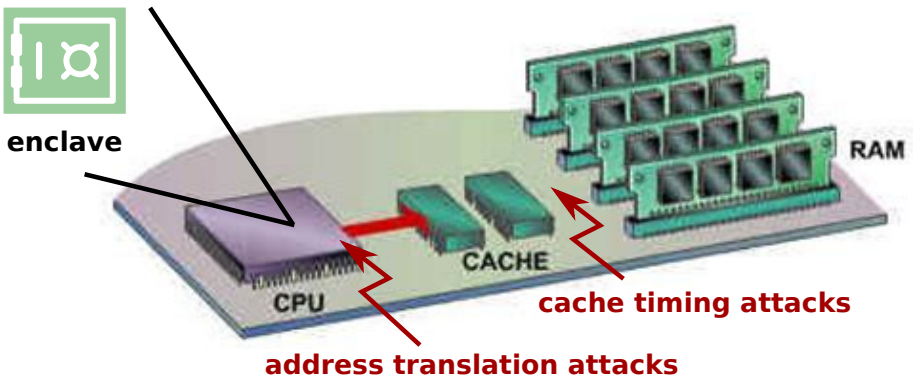
```
1 int secret_lookup(int s)
2 {
3     if (s > 0 && s < ARRAY_LEN)
4         return array[s];
5     return -1;
6
7 }
```

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

Overview: Spying on enclave memory accesses



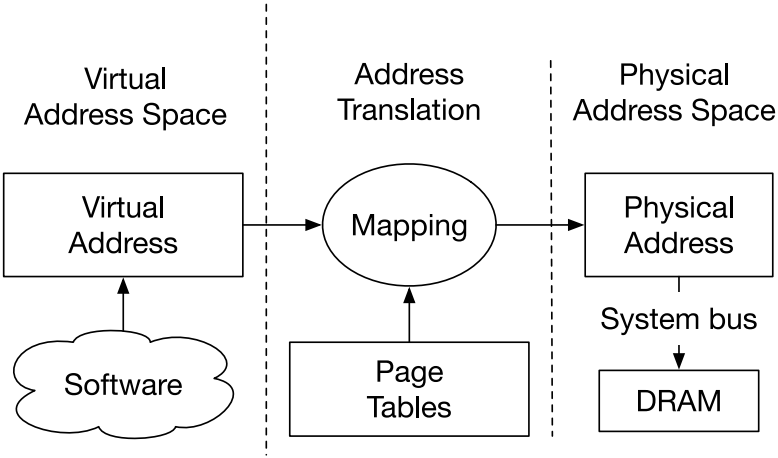
Overview: Spying on enclave memory accesses



1. Privileged Side-Channel Attacks

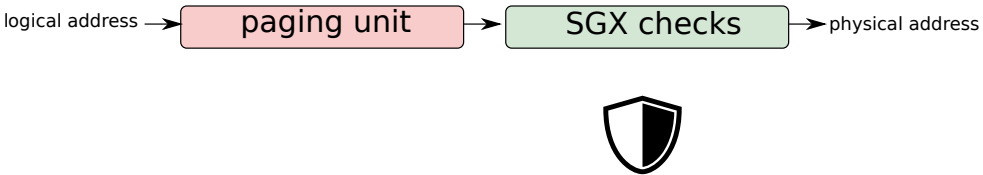
Idea #1: Monitoring Address Translation?

The virtual memory abstraction



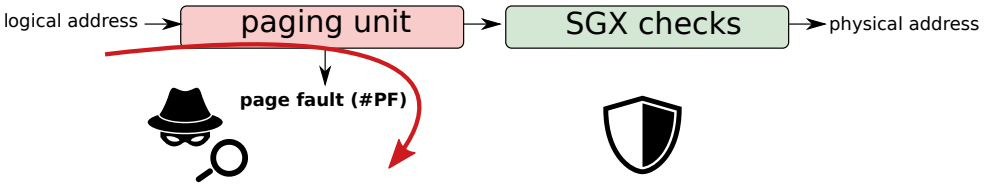
Costan et al. "Intel SGX explained", IACR 2016.

Intel SGX: Page faults as a side channel



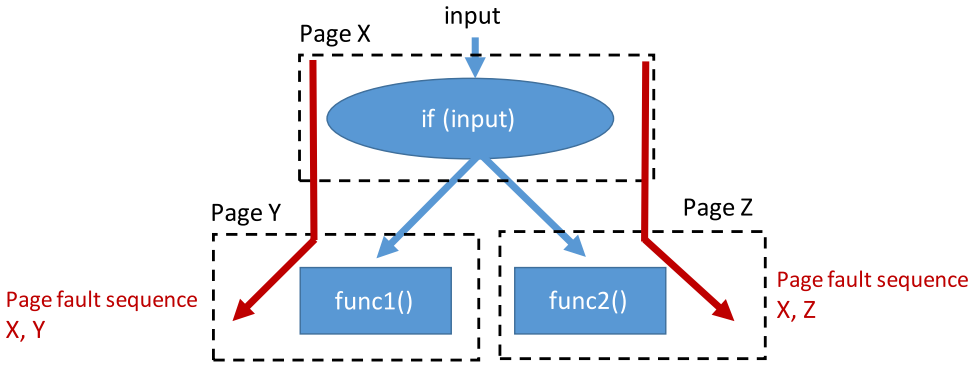
SGX machinery protects against direct address remapping attacks

Intel SGX: Page faults as a side channel



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

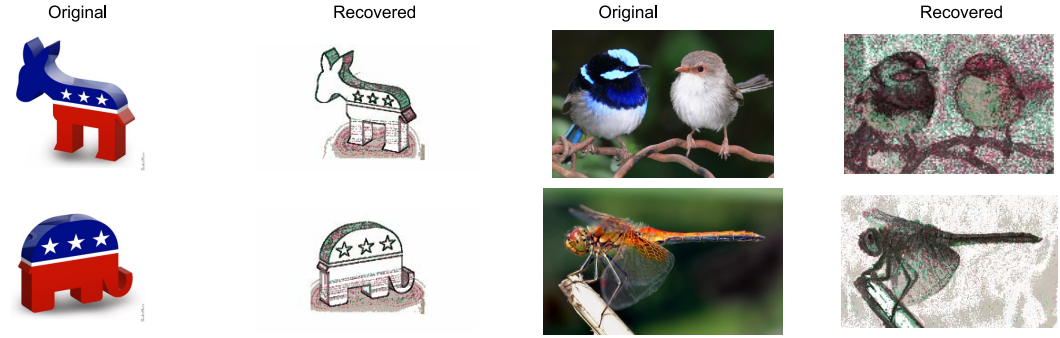
Intel SGX: Page faults as a side channel



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

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

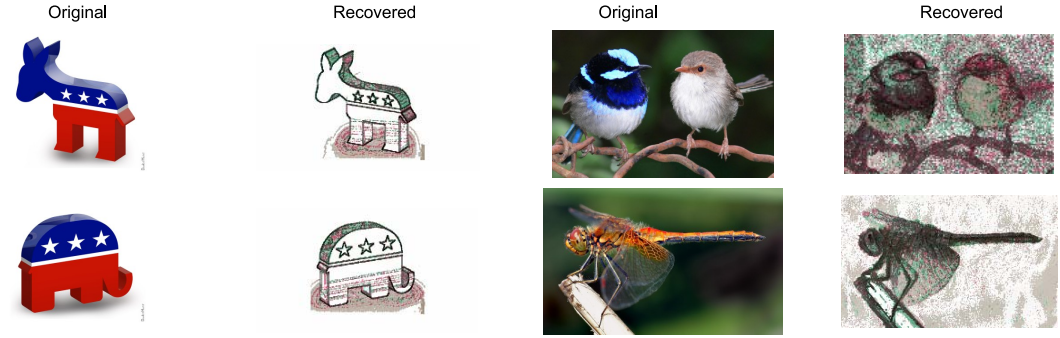
Page table-based attacks in practice



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



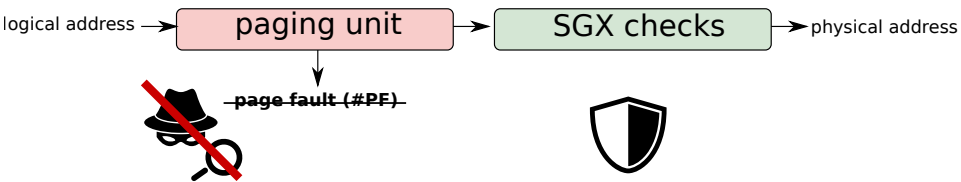
Xu et al.: "Controlled-channel attacks: Deterministic side channels for untrusted operating systems", Oakland 2015.

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

1. Privileged Side-Channel Attacks

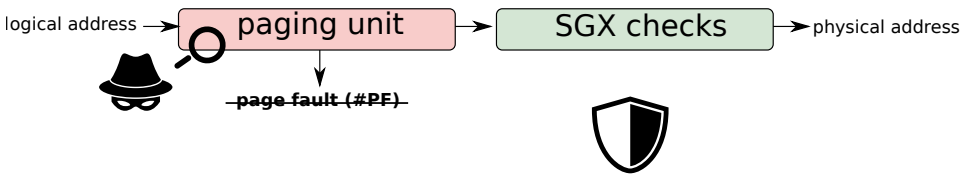
Idea #2: Monitoring without Page Faults?

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.

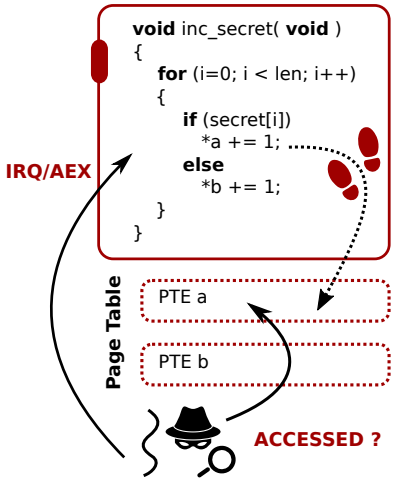
Whenever the processor uses a paging-structure entry as part of linear-address translation, it sets the accessed flag in that entry (if it is not already set).

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.

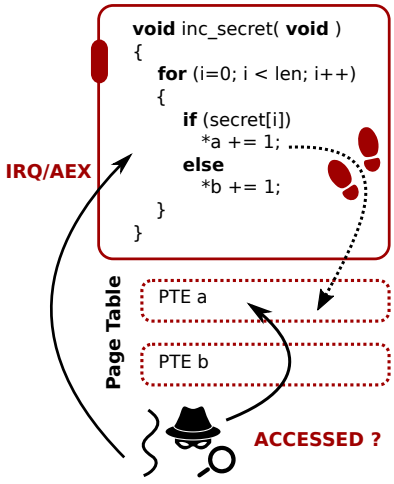
Telling your secrets without page faults

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!

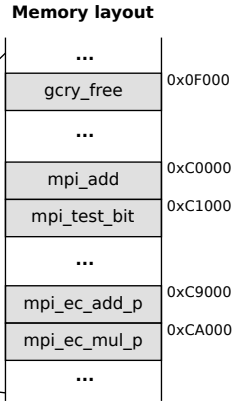


Van Bulck et al. "Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution", USENIX 2017.

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 }
```

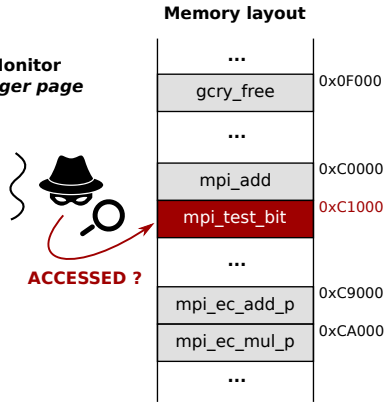
22 Code pages per iteration



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 }
```



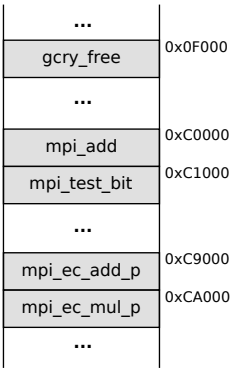
Van Bulck et al. "Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution", USENIX 2017.

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

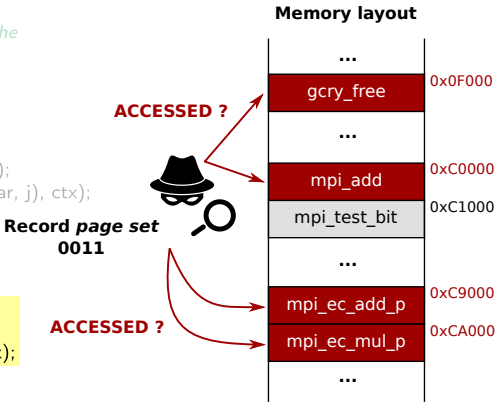
Memory layout



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.

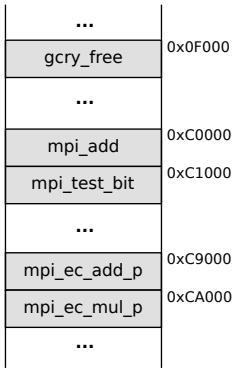
Attacking Libgcrypt EdDSA (simplified)

```
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

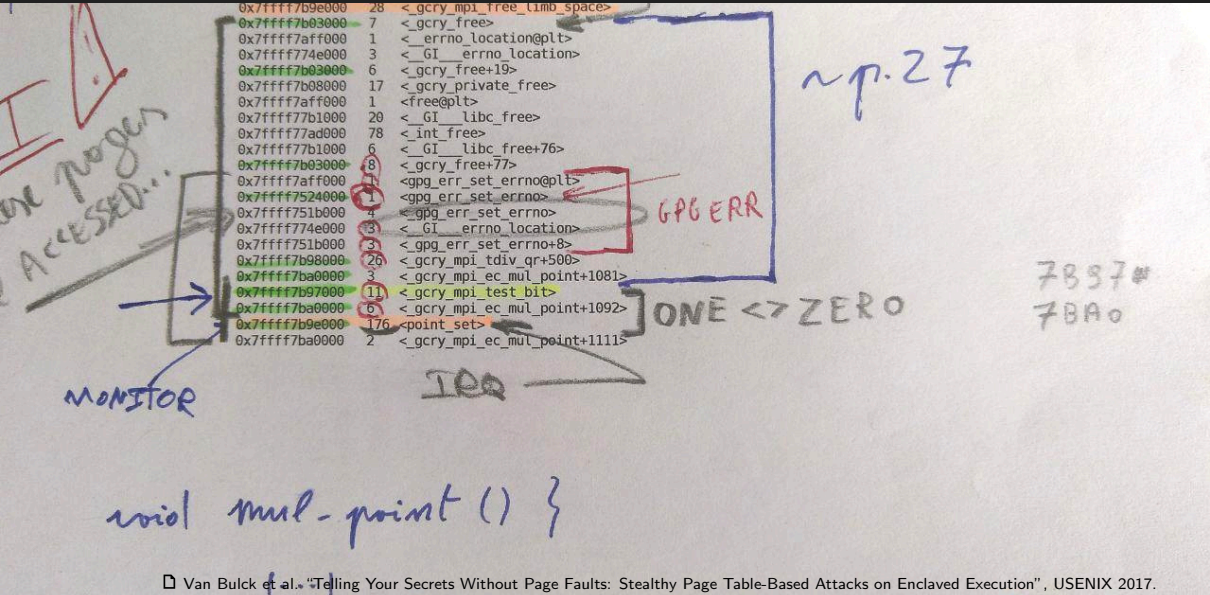
Full 512-bit key recovery, single run

Memory layout



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

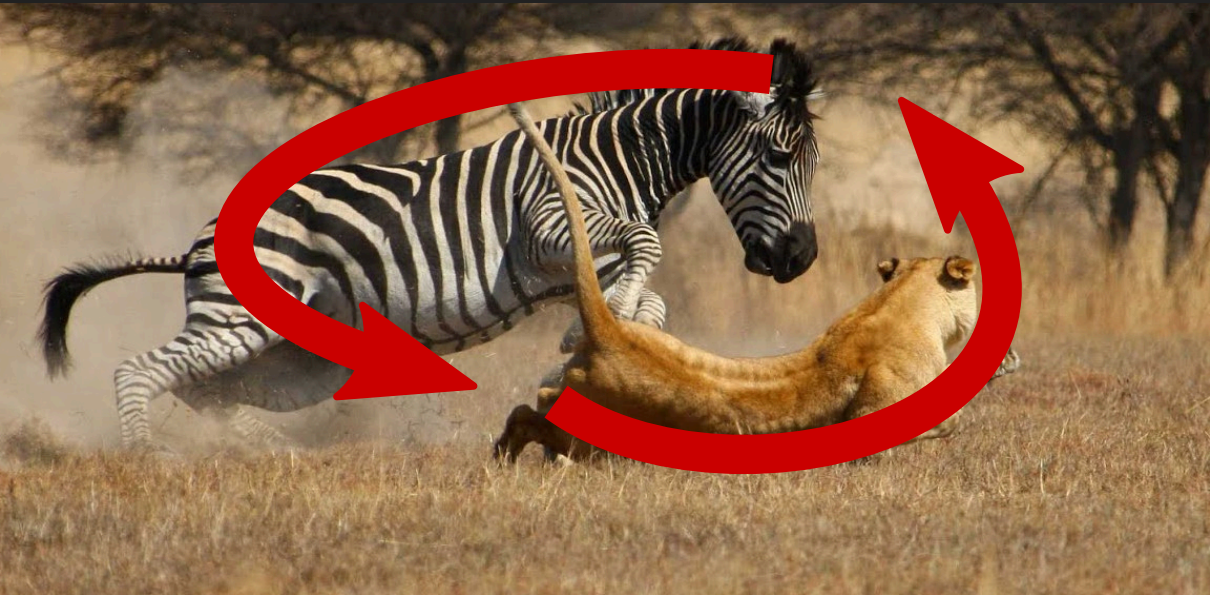
Side-channel analysis: From metadata patterns to secrets



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

```
1  size_t strlen (char *str)
2  {
3    char *s;
4
5    for (s = str; *s; ++s);
6    return (s - str);
7  }
```

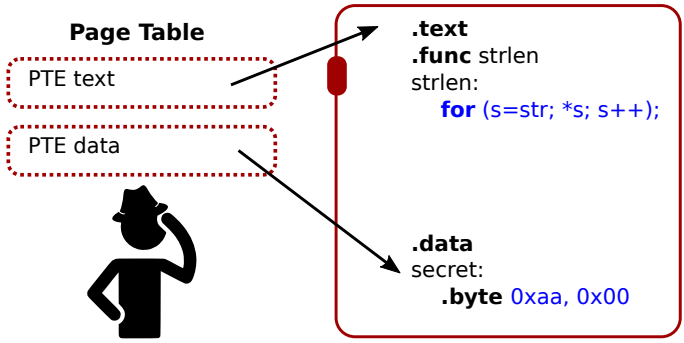
```
1      mov  %rdi,%rax
2  1:  cmpb $0x0,(%rax)
3      je   2f
4      inc  %rax
5      jmp  1b
6  2:  sub  %rdi,%rax
7      retq
```

⇒ 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

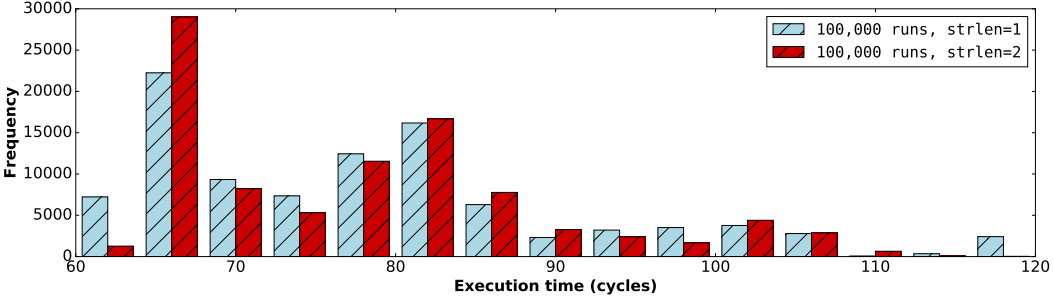
Temporal resolution limitations for the page fault oracle



Counting `strlen` loop iterations?

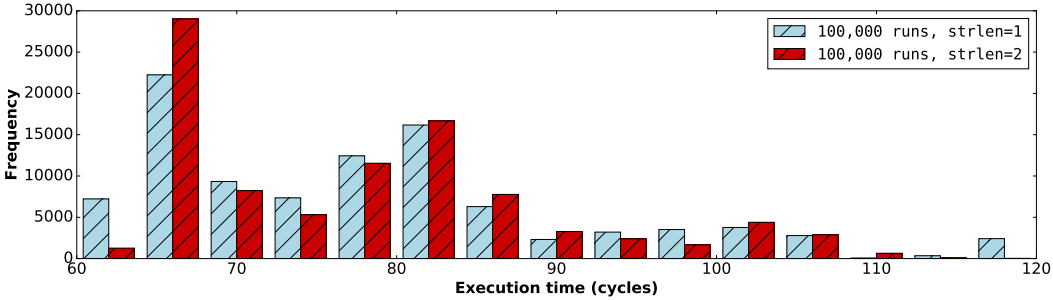
Progress requires both pages present (non-faulting) ↔ page fault oracle

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...



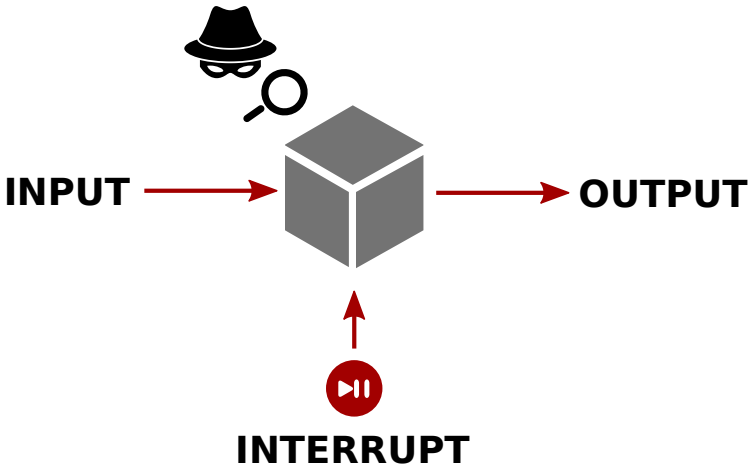
Challenge: Side-channel Sampling Rate

Slow
shutter speed

Medium
shutter speed

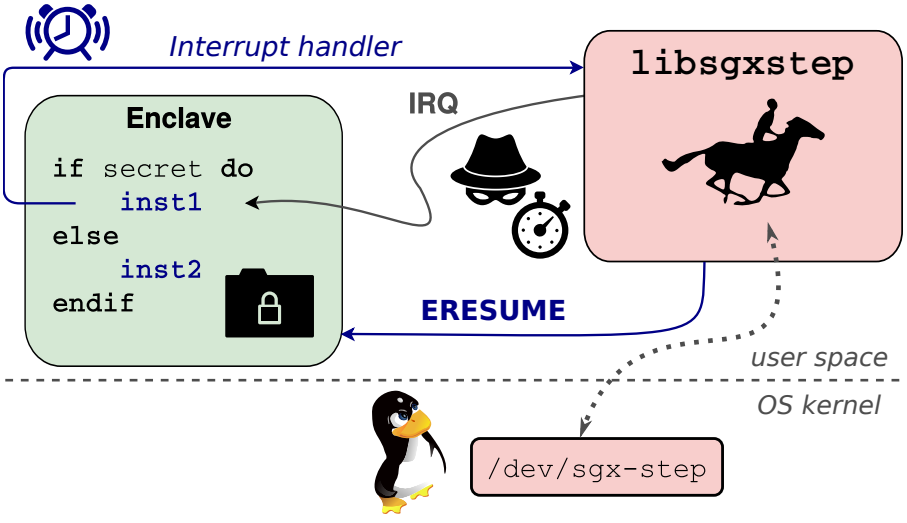
Fast
shutter speed

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

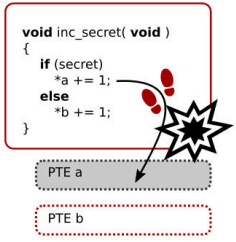
```
[idt.c] DTR.base=0xfffffe0000000000/size=4095 (256 entries)
[idt.c] established user space IDT mapping at 0x7f7ff8e9a000
[idt.c] installed asm IRQ handler at 10:0x56312d19b000
[idt.c] IDT[ 45] @0x7f7ff8e9a2d0 = 0x56312d19b000 (seg sel 0x10); p=1; dpl=3; type=14; ist=0
[file.c] reading buffer from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

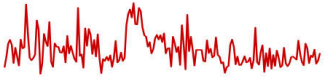
SGX-Step: Enabling a New Line of High-Resolution Attacks

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)	✓

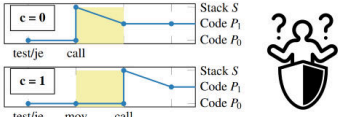
A Versatile Open-Source Attack Toolkit

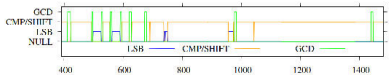




Interrupt latency

[CCS'18, USENIX'21]





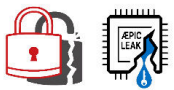
Page-table manipulation

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

SGX-Step

Interrupt counting

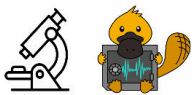
[CCS'19, CHES'20-21, USENIX'20]



High-resolution probing

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

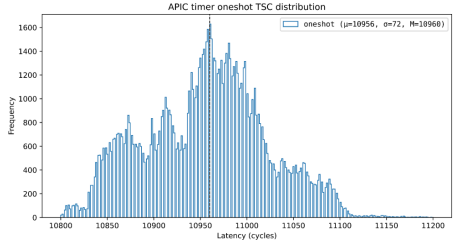
[USENIX'18, CCS'19, S&P'21] Zero-step replaying

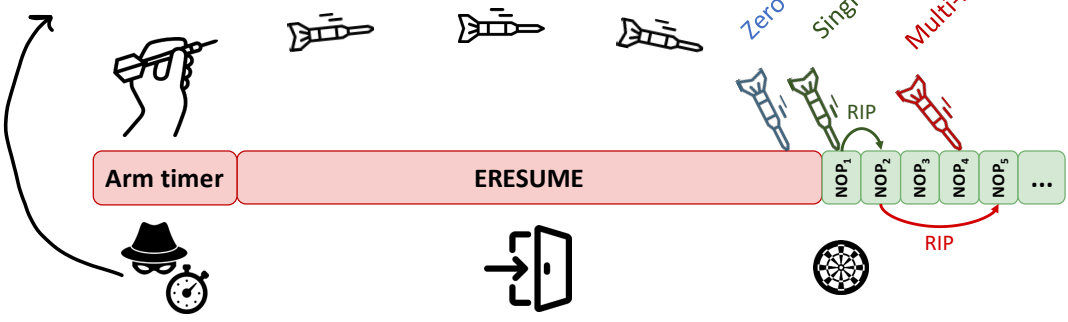


1. Privileged Side-Channel Attacks

Idea #4: Temporal 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

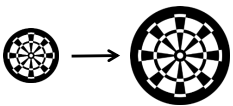
3. Assisted PT walk

1. Clear PTE A-bit

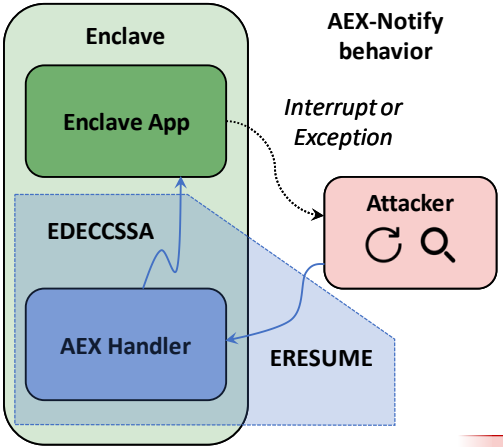
2. TLB flush

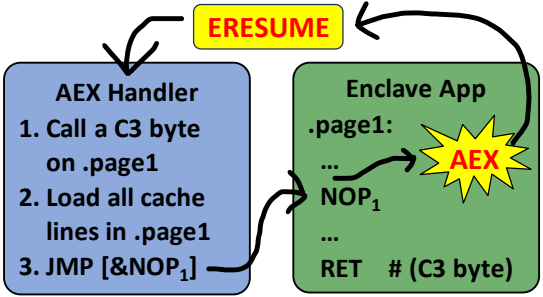
page walk ($RIP) | *exec*

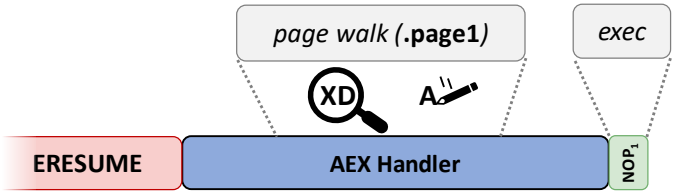
Arm timer | **ERESUME** | **NOP$_1$**



AEX-Notify solution overview



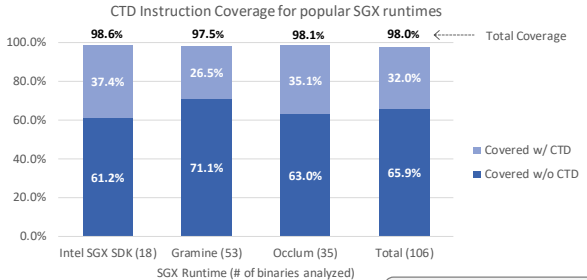


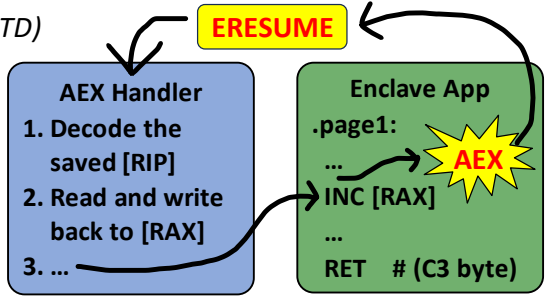


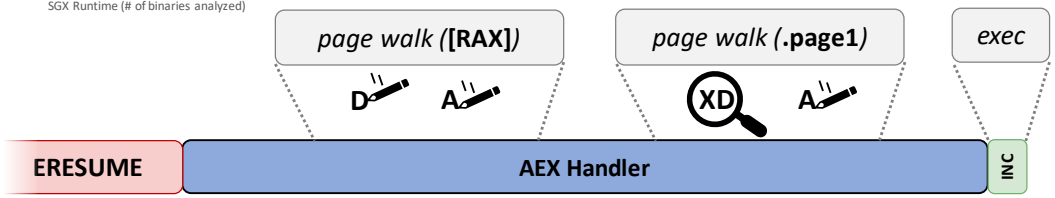
Legend: AEX-Notify ISA Extension

AEX-Notify solution overview

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







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**

1. Privileged Side-Channel Attacks

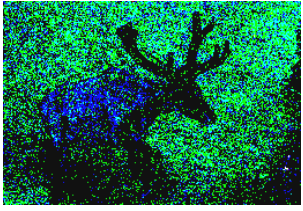
Idea #5: Spatial Hardening?

There's a Catch...

Finally note that our proposed mitigation does not protect against interrupting enclaves and observing application code and data page accesses at a coarse-grained 4 KiB spatial resolution. In contrast to the fine-grained, instruction-granular interrupt-driven attacks we consider in this work, such controlled-channel attacks have received ample attention [18, 47, 56, 59] from the research community.

Constable et al. "AEX-Notify: Thwarting Precise Single-Stepping Attacks through Interrupt Awareness for Intel SGX Enclaves", USENIX Security 23.

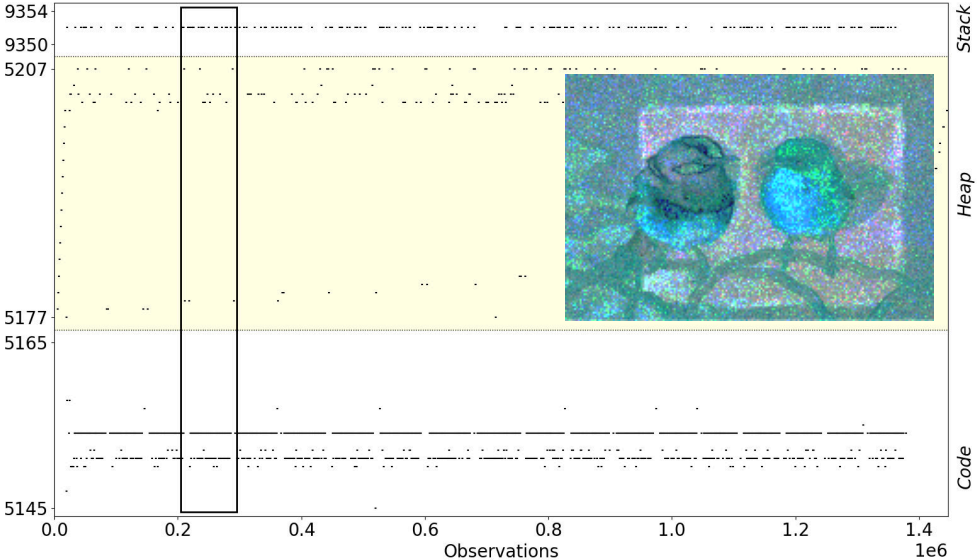
Why Mitigating Single-Stepping is Not Enough



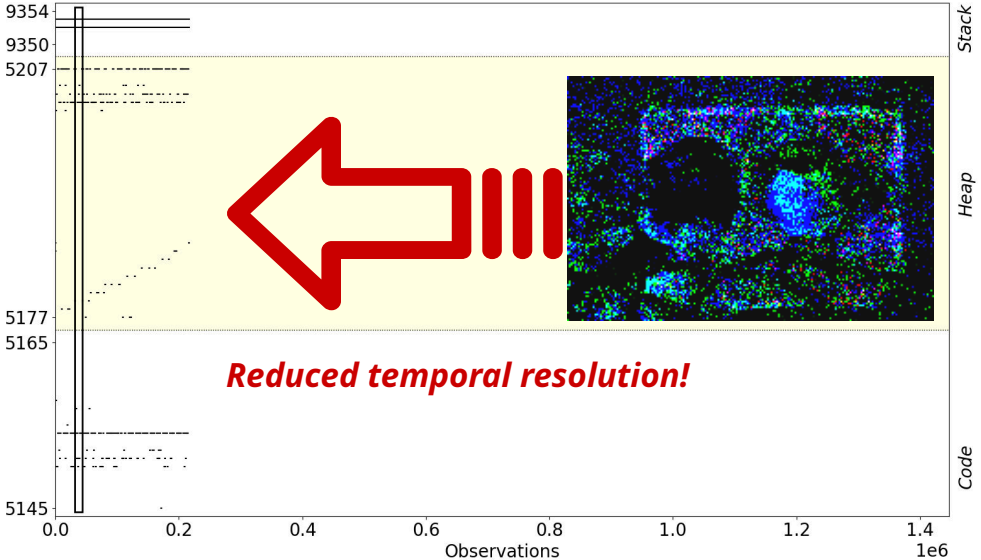
Original (left), Xu et al. (middle), our attack with AEX-Notify single-stepping defense (right)

- Xu et al.: "Controlled-channel attacks: Deterministic side channels for untrusted operating systems", Oakland 2015.
- Constable et al.: "AEX-Notify: Thwarting Precise Single-Stepping Attacks through Interrupt Awareness for Intel SGX Enclaves", USENIX Security 2023.

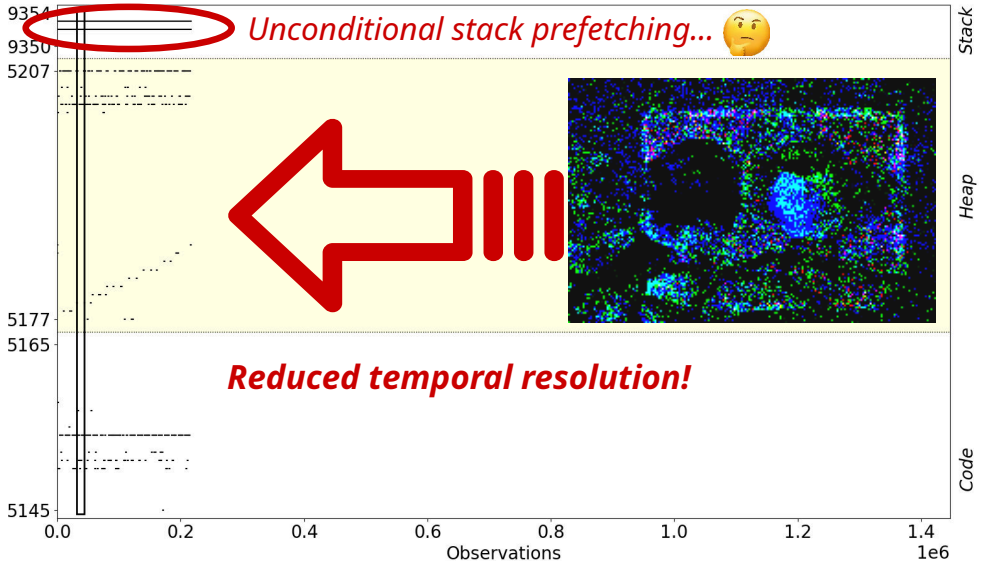
Libjpeg: AEX-Notify's Temporal Reduction in Practice



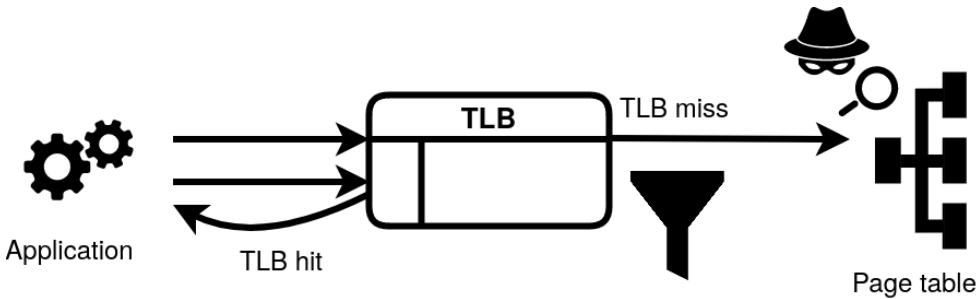
Libjpeg: AEX-Notify's Temporal Reduction in Practice



Libjpeg: AEX-Notify's Temporal Reduction in Practice

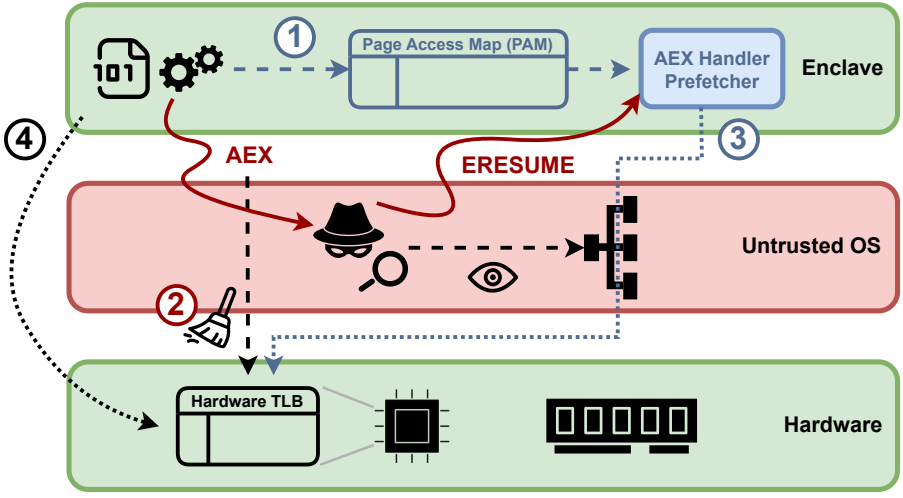


Idea: TLB as a “Filter” to Hide Page Accesses

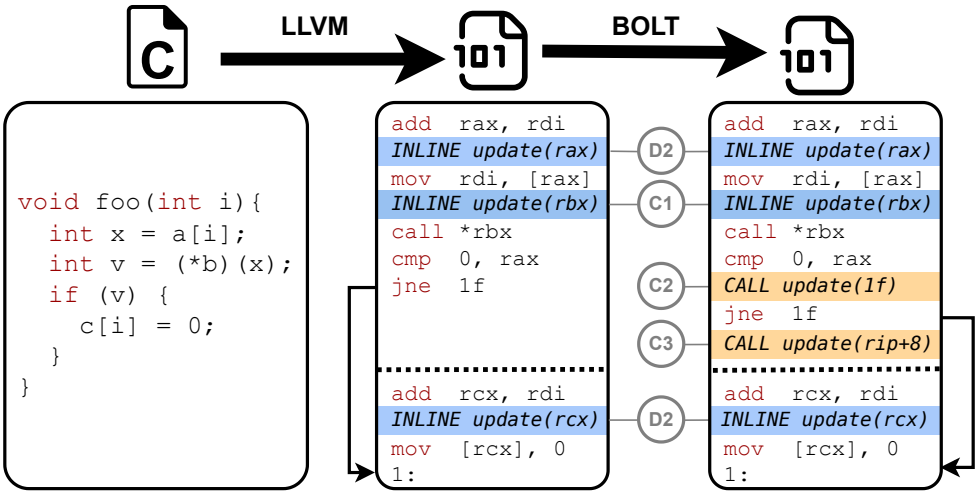


🗋 Vanoverloop et al.: “TLBlur: Compiler-Assisted Automated Hardening against Controlled Channels on Off-the-Shelf Intel SGX Platforms”, USENIX Security 2025.

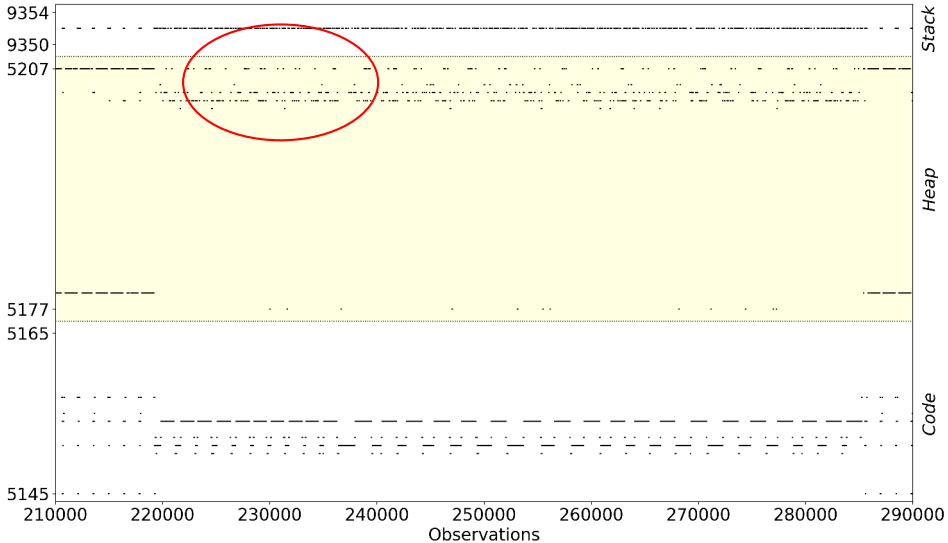
TLBlur: Self-Monitoring and Restoring Enclave Page Accesses



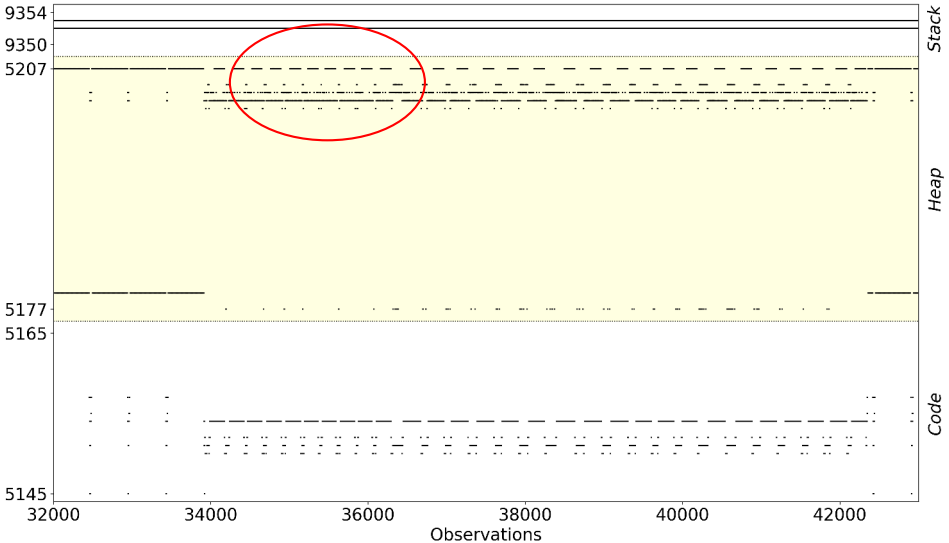
Instrumentation to Self-Monitor Page Accesses at Runtime



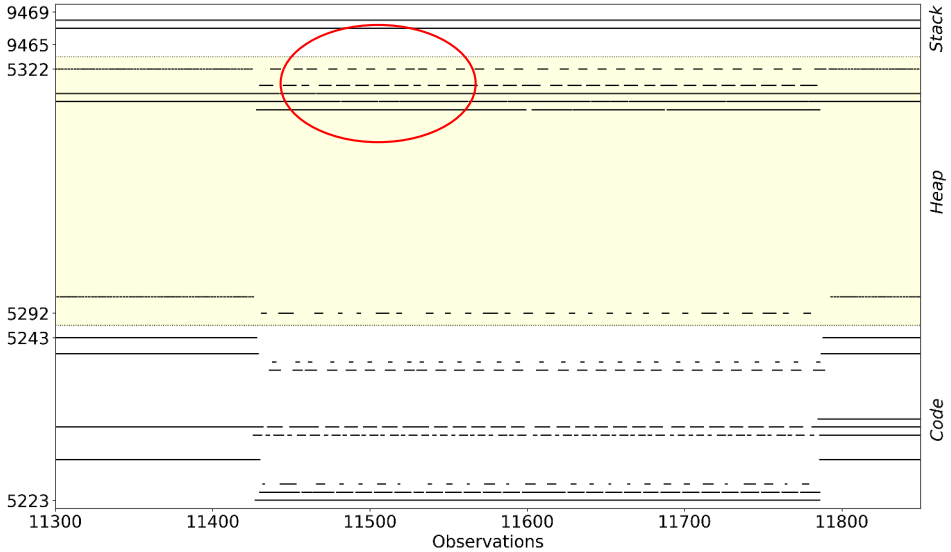
Leakage Reduction in Practice: libjpeg with Single Stepping



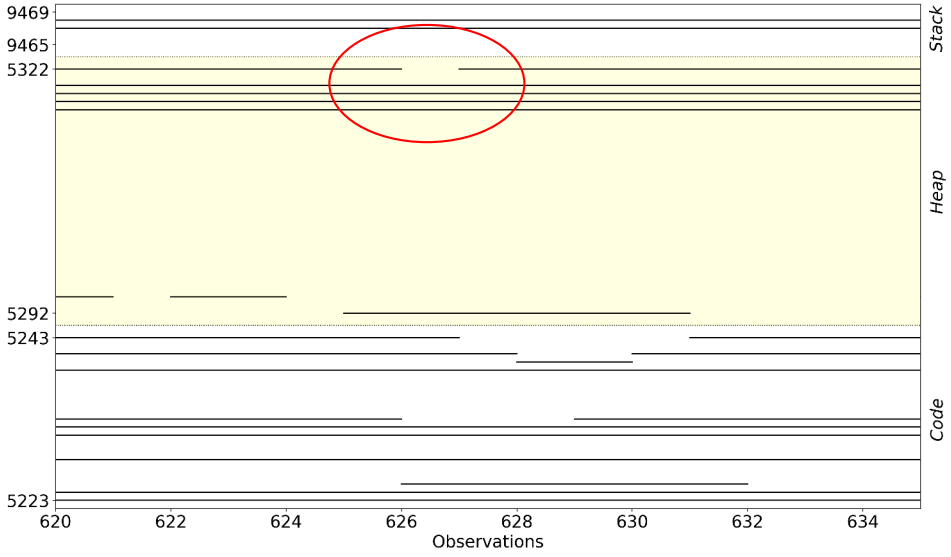
Leakage Reduction in Practice: libjpeg with Page Faults



Leakage Reduction in Practice: libjpeg with TLBlur ($N = 10$)



Leakage Reduction in Practice: libjpeg with TLBlur ($N = 20$)



Leakage Reduction in Practice: libjpeg with TLBlur ($N = 30$)

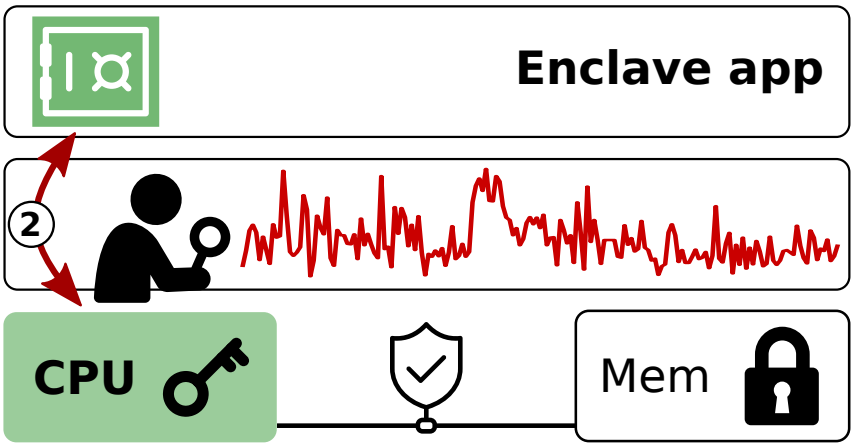
2. Transient-Execution Attacks

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

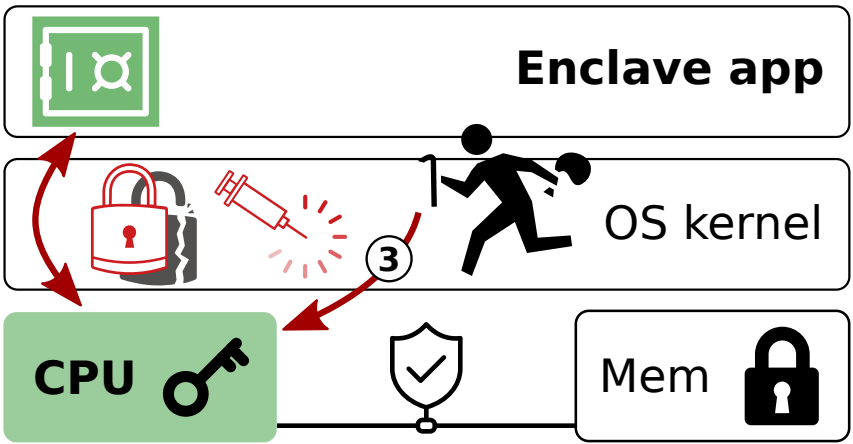
"In general, these research papers do not demonstrate anything new or unexpected about the Intel SGX architecture. Preventing side channel attacks is a matter for the enclave developer. *Intel makes this clear in the security objectives for Intel SGX."*

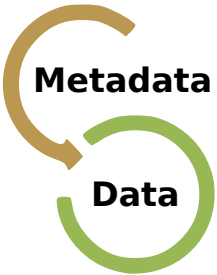
https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sgx-and-side-channels.html

Recap: Privileged side-channel attacks



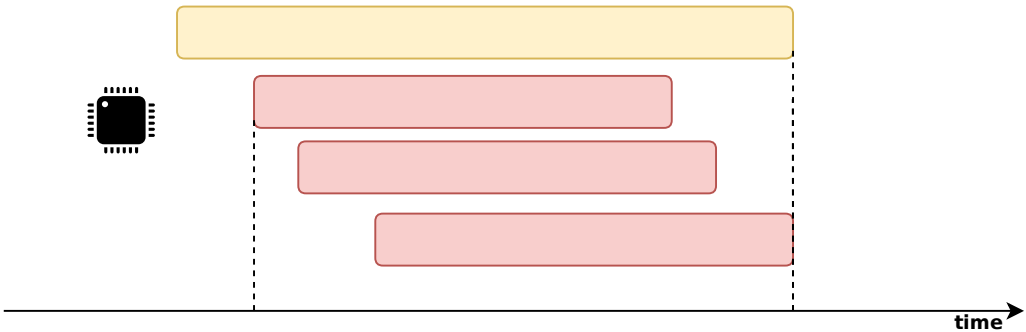
Recap: Transient-execution attacks



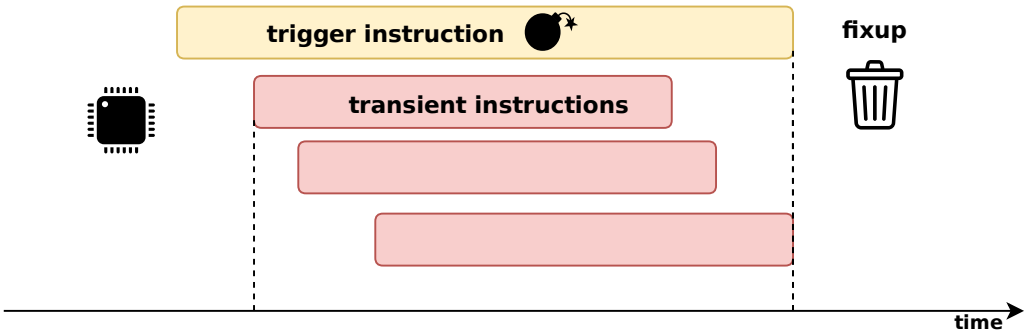



WHAT IF I TOLD YOU
YOU CAN CHANGE RULES MID-GAME
imgflip.com

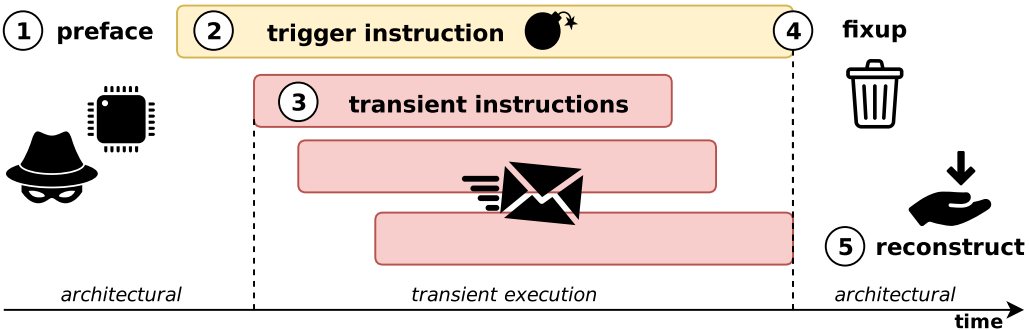
Abusing out-of-order and speculative execution



Abusing out-of-order and speculative execution



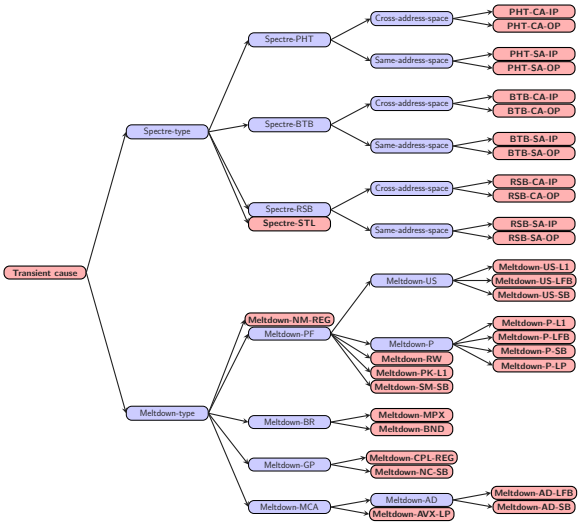
Abusing out-of-order and speculative execution

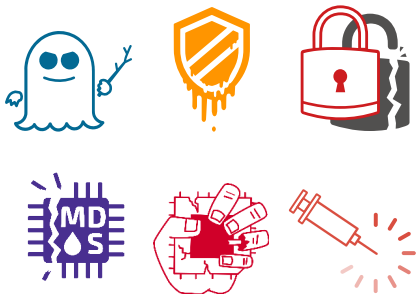


Transient-execution attacks: Welcome to the world of fun!

The transient-execution zoo

https://transient.fail

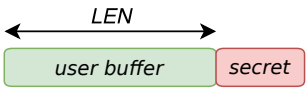




2. Transient-Execution Attacks

Confused-deputy code gadgets?

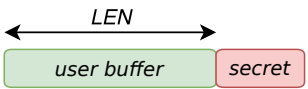
Spectre v1: Speculative buffer over-read




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

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

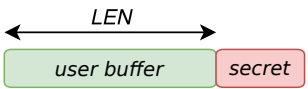
Spectre v1: Speculative buffer over-read




```
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

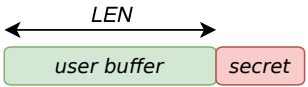
Spectre v1: Speculative buffer over-read




```
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
- **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

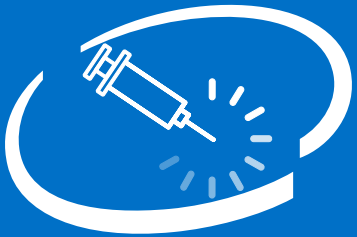
- CPU **transiently** executes wrong code paths
- **Confused-deputy gadgets** encode secrets via side channels

2. Transient-Execution Attacks

Transient access-control bypass?

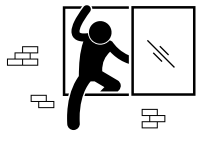
inside™

inside™



inside™

Meltdown: Transiently encoding unauthorized memory



Unauthorized access

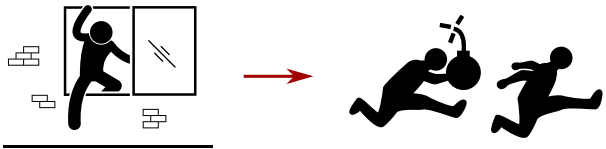
Listing 1: x86 assembly

```
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

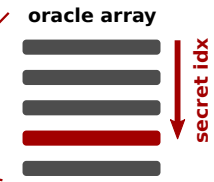
Transient out-of-order window

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)

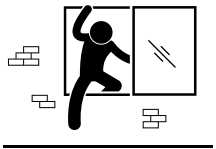
Listing 1: x86 assembly.

```
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

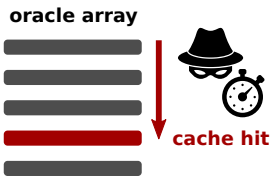
Exception handler

Listing 1: x86 assembly.

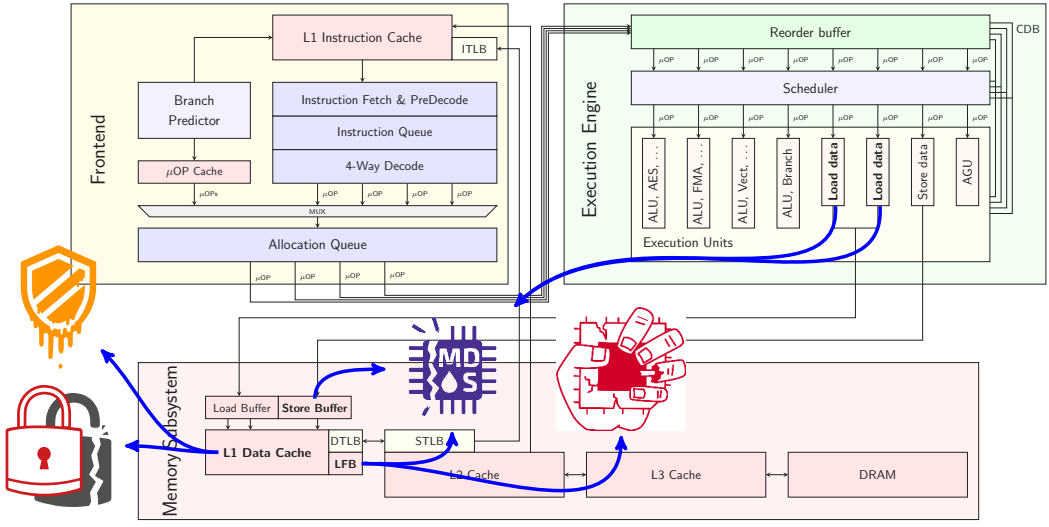
```
1 meltdown:
2   // %rdi: oracle
3   // %rsi: secret_ptr
4
5   movb (%rsi), %al
6   shl $0xc, %rax
7   movq (%rdi, %rax), %rdi
8   retq
```

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 }
```



MDS variants: Flushing additional microarchitectural buffers



Meltdown take-away

- **Faulting** loads transiently forward data from various microarchitectural buffers
- **Hardware fixes** for recent CPUs

Conclusions and Outlook

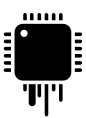
DALL·E 3

Takeaways

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

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

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



Thank you! Questions?