Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution

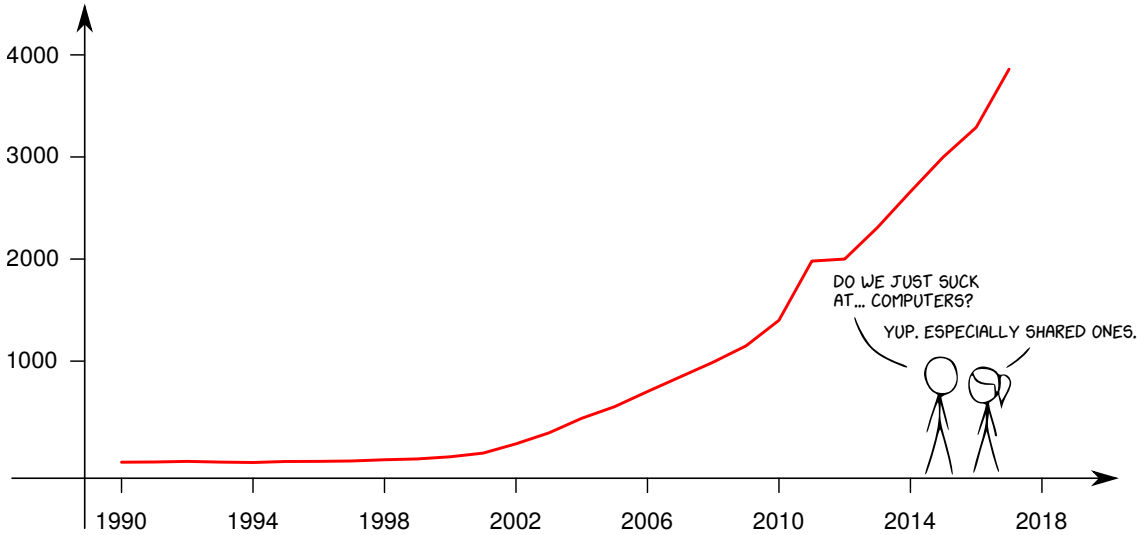
Jo Van Bulck [1] Marina Minkin [2] Ofir Weisse [3] Daniel Genkin [3] Baris Kasikci [3]
Frank Piessens [1] Mark Silberstein [2] Thomas F. Wenisch [3] Yuval Yarom [4] Raoul Strackx [1]

[1]imec-DistriNet, KU Leuven [2]Technion [3]University of Michigan [4]University of Adelaide and Data61

USENIX Security, August 2018

Road map

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

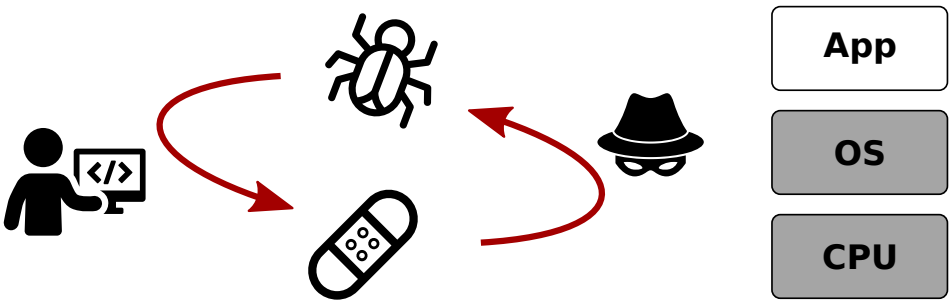


Based on github.com/Pold87/academic-keyword-occurrence and xkcd.com/1938/

Security in a post-Meltdown world

Classic attacker-defender race

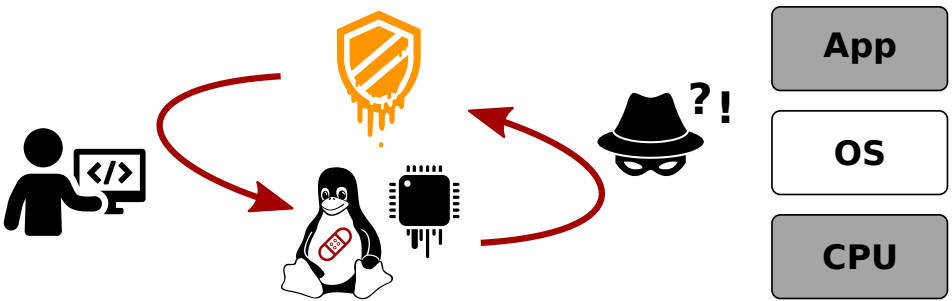
Exploit and patch **application-level** vulnerabilities (memory safety, side-channels)



Security in a post-Meltdown world

Game changer Meltdown

Free universal read primitive → kernel page-table isolation



Rumors: Meltdown immunity for SGX enclaves?

Meltdown melted down everything, except for one thing

"[enclaves] remain protected and completely secure"

— *International Business Times, February 2018*

ANJUNA'S SECURE-RUNTIME CAN PROTECT CRITICAL APPLICATIONS AGAINST THE MELTDOWN ATTACK USING ENCLAVES

"[enclave memory accesses] redirected to an abort page, which has no value"

— *Anjuna Security, Inc., March 2018*

~~Rumors: Meltdown immunity for SGX enclaves?~~

LILY HAY NEWMAN SECURITY 08.14.18 01:00 PM

SPECTRE-LIKE FLAW UNDERMINES INTEL PROCESSORS' MOST SECURE ELEMENT

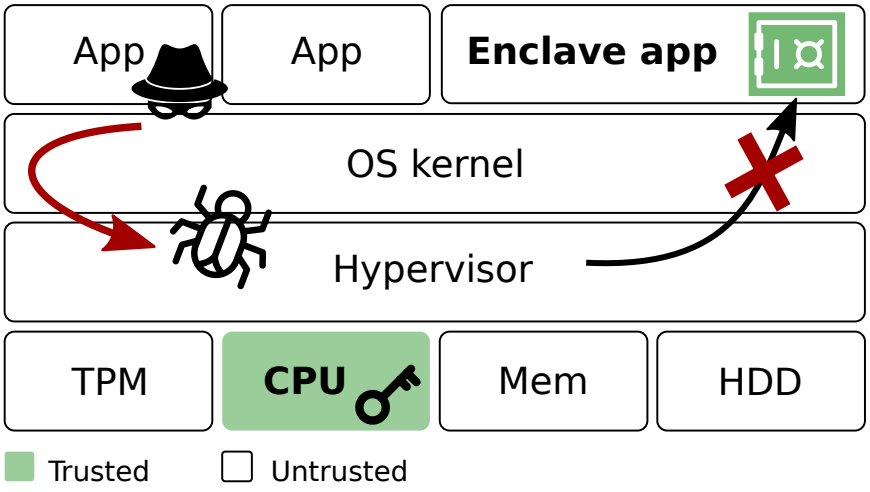
I'M SURE THIS WON'T BE THE LAST SUCH PROBLEM —

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

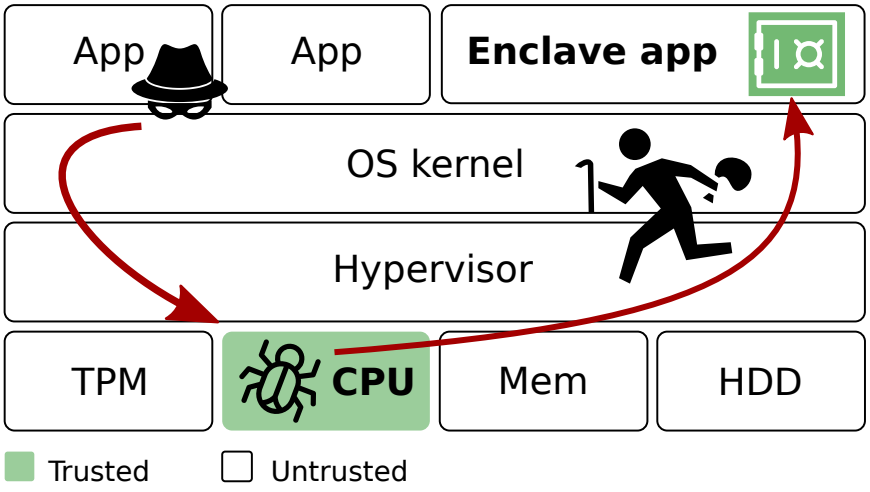
Speculative execution attacks truly are the gift that keeps on giving.

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

Intel SGX promise: Hardware-level isolation and attestation



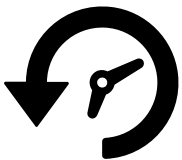
Intel SGX promise: Hardware-level isolation and attestation




FAIL

Road map

Building Foreshadow

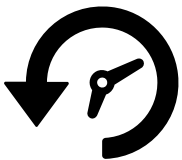


1. Cache secrets in L1

2. Unmap page table entry

3. Execute Meltdown

Building Foreshadow



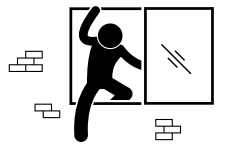
1. Cache secrets in L1

2. Unmap page table entry

3. Execute Meltdown

L1 terminal fault challenges

Meltdown recap: Transiently encoding unauthorized memory



Unauthorized access

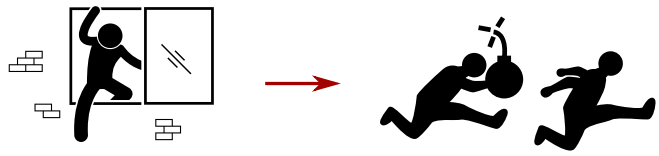
Listing 1: x86 assembly

```
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 recap: Transiently encoding unauthorized memory



Unauthorized access

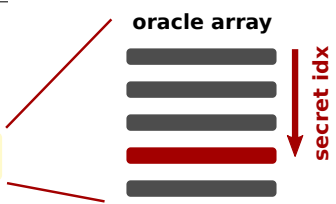
Transient out-of-order window

Listing 1: x86 assembly.

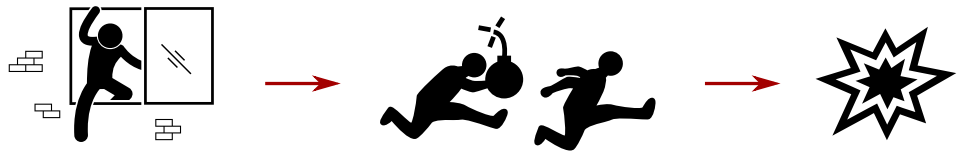
```
1  meltdown:
2    // %rdi: oracle
3    // %rsi: secret_ptr
4
5    movb (%rsi), %al
6    shl $0xc, %rax
7    movq (%rdi, %rax), %rdi
8    retq
```

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 recap: Transiently encoding unauthorized memory



Unauthorized access

Transient out-of-order window

Exception
(discard architectural state)

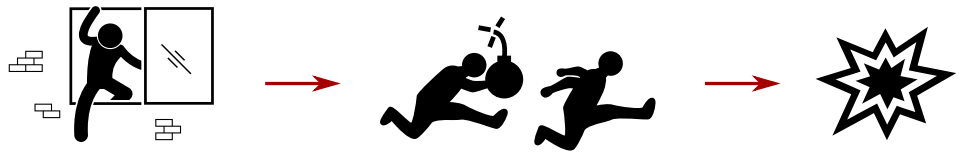
Listing 1: x86 assembly.

```
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 recap: Transiently encoding unauthorized memory



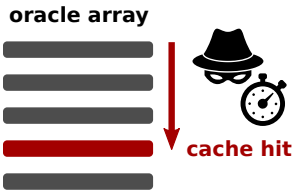
Unauthorized access | Transient out-of-order window | **Exception handler**

Listing 1: x86 assembly.

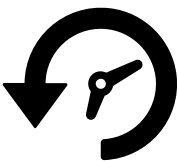
```
1 meltdown:
2     // %rdi: oracle
3     // %rsi: secret_ptr
4
5     movb (%rsi), %al
6     shl $0xc, %rax
7     movq (%rdi, %rax), %rdi
8     retq
```

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 }
```



Challenge #1: Intel SGX abort page semantics



1. Cache secrets in L1

2. Unmap page table entry

3. Execute Meltdown

Challenge #1: Intel SGX abort page semantics

Untrusted world view

- Enclaved memory reads `0xFF`

Intra-enclave view

- Access enclaved + unprotected memory

Challenge #1: Intel SGX abort page semantics

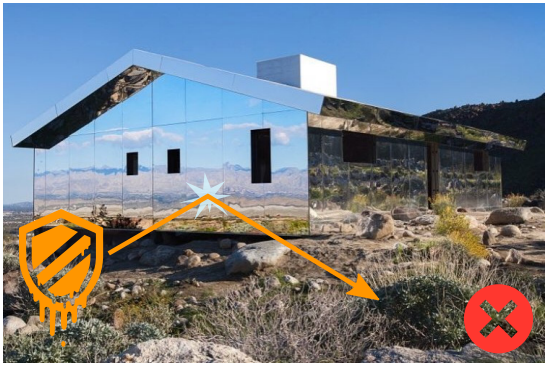
Untrusted world view

- Enclaved memory reads `0xFF`

Intra-enclave view

- Access enclaved + unprotected memory
- SGXpectre in-enclave code abuse

Challenge #1: Intel SGX abort page semantics



Untrusted world view

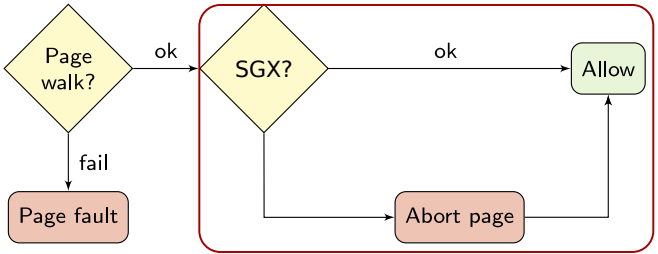
- Enclaved memory reads `0xFF`
- Meltdown "bounces back" ($\sim$ mirror)

Intra-enclave view

- Access enclaved + unprotected memory
- SGXpectre in-enclave code abuse

Building Foreshadow: Evade the abort page

Note: SGX MMU sanitizes *untrusted* address translation

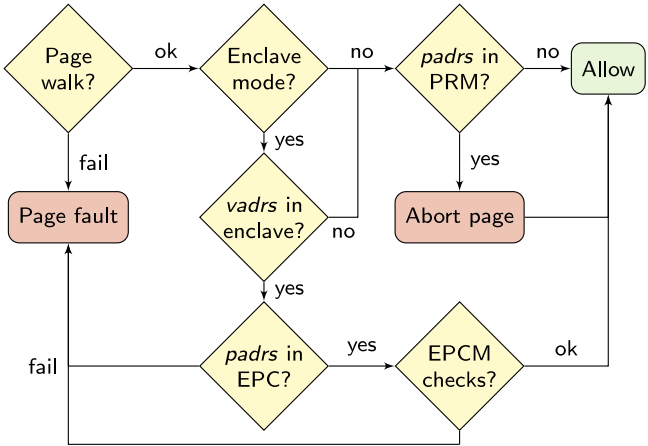


Abort page semantics:
An attempt to read from a non-existent or disallowed resource returns all ones for data (abort page). An attempt to write to a non-existent or disallowed physical resource is dropped. This behavior is unrelated to exception type abort (the others being Fault and Trap).

https://software.intel.com/en-us/sgx-sdk-dev-reference-enclave-development-basics

Building Foreshadow: Evade the abort page

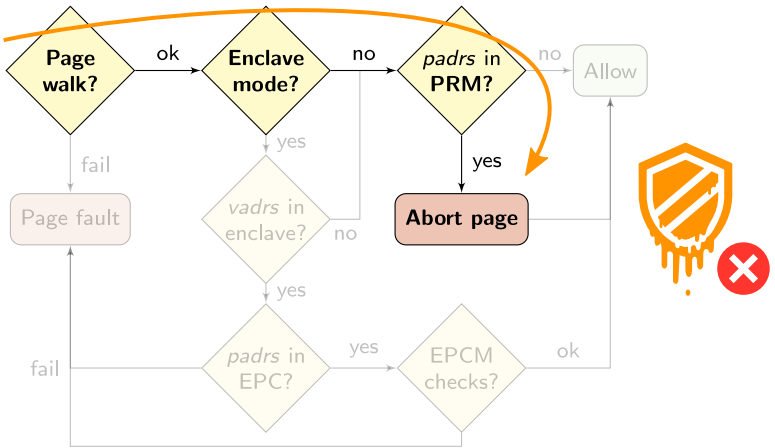
Note: SGX MMU sanitizes *untrusted* address translation



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

Building Foreshadow: Evade the abort page

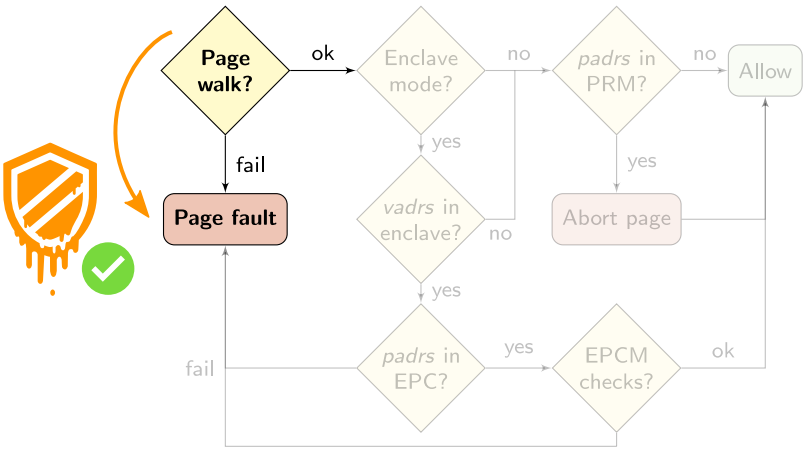
Straw man: (Speculative) accesses in non-enclave mode are dropped



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

Building Foreshadow: Evade the abort page

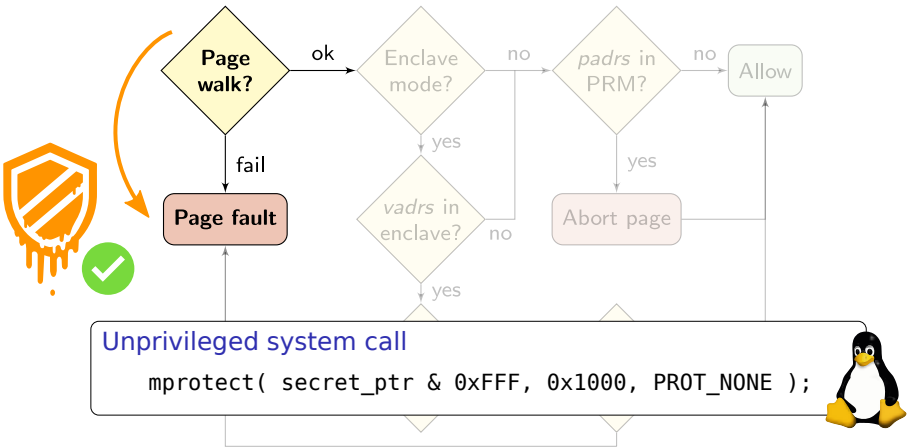
Stone man: Bypass abort page via *untrusted* page table



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

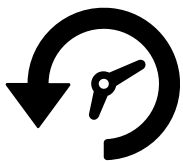
Building Foreshadow: Evade the abort page

Stone man: Bypass abort page via *untrusted* page table



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

Challenge #2: Strict caching requirements



1. Cache secrets in L1

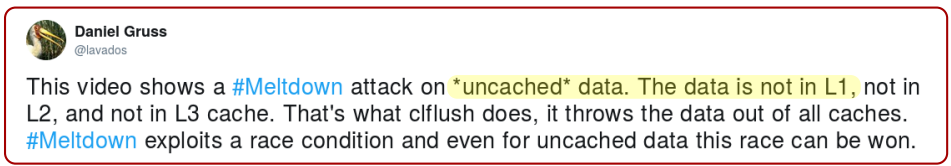
2. Unmap page table entry

3. Execute Meltdown

Challenge #2: Strict caching requirements

L1 terminal fault

Only **enclave loads served from L1** reach transient out-of-order execution



https://twitter.com/lavados/status/951066835310534656

Challenge #2: Strict caching requirements

L1 terminal fault

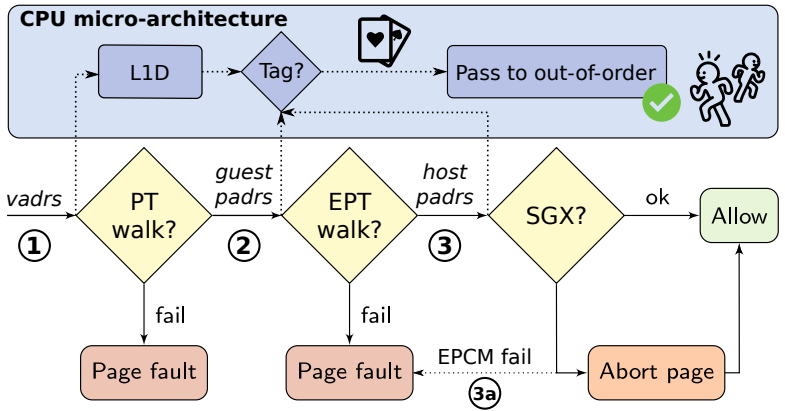
Only **enclave loads served from L1** reach transient out-of-order execution

Foreshadow present bit ↔ Meltdown supervisor bit

Challenge #2: Strict caching requirements

Intel micro-architecture

Address translation abort *in parallel* with L1 lookup (tag comparison)



Weisse et al. "Foreshadow-NG: Breaking the Virtual Memory Abstraction with Transient Out-of-Order Execution"

Building Foreshadow: Loading enclave secrets in L1

1. Preemptive extraction

   Interrupt victim enclave at page or instruction-level granularity

   → *Memory operands + CPU registers (SSA)*

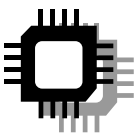
Van Bulck et al. "SGX-Step: A practical attack framework for precise enclave execution control", SysTEX 2017

Building Foreshadow: Loading enclave secrets in L1

1. Preemptive extraction

   Interrupt victim enclave at page or instruction-level granularity

   → *Memory operands + CPU registers (SSA)*



2. Concurrent extraction

   Intel **HyperThreading:** co-resident logical CPUs share L1

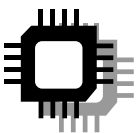
   → *Real time memory accesses*

Building Foreshadow: Loading enclave secrets in L1

1. Preemptive extraction

   Interrupt victim enclave at page or instruction-level granularity

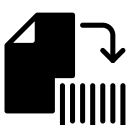
   → *Memory operands + CPU registers (SSA)*



2. Concurrent extraction

   Intel **HyperThreading:** co-resident logical CPUs share L1

   → *Real time memory accesses*

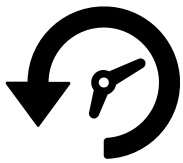


3. Uncached extraction

   Forcibly reload 4 KiB enclave **page:** `ewb` + `eldu`

   → *Reliably dump entire enclave address space*

Building Foreshadow: Loading enclave secrets in L1



1. Cache secrets in L1

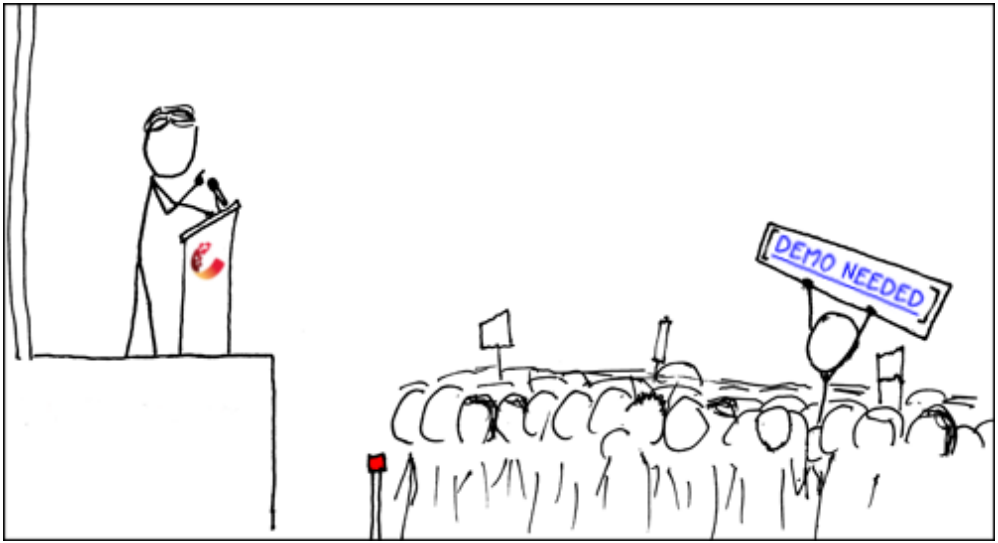
2. Unmap page table entry

3. Execute Meltdown

Many more **optimization techniques + microbenchmarks** → see paper!

Road map

Demo time!



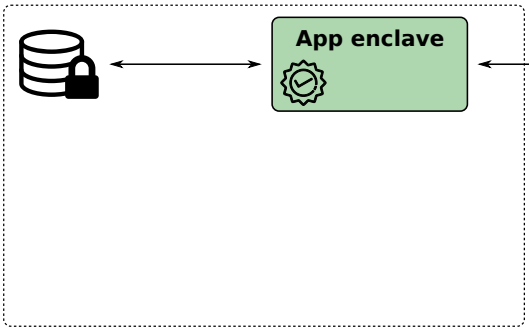
Based on xkcd.com/285/

Road map

Establishing trust: Remote attestation and secret provisioning

Binding secrets to enclave identity

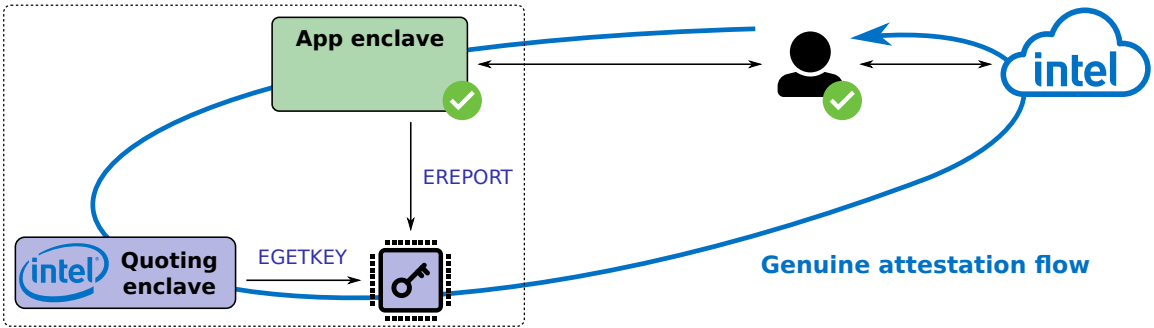
Goal: Secure end-to-end **communication** channel + local **storage**



Establishing trust: Remote attestation and secret provisioning

CPU-level key derivation

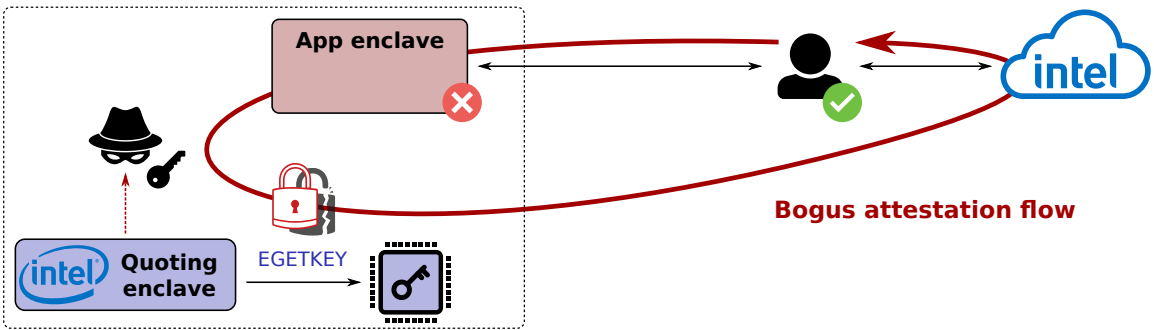
Intel == trusted 3th party (shared **CPU master secret**)



Eroding trust: ~~Remote attestation~~ and secret provisioning

Foreshadow adversary

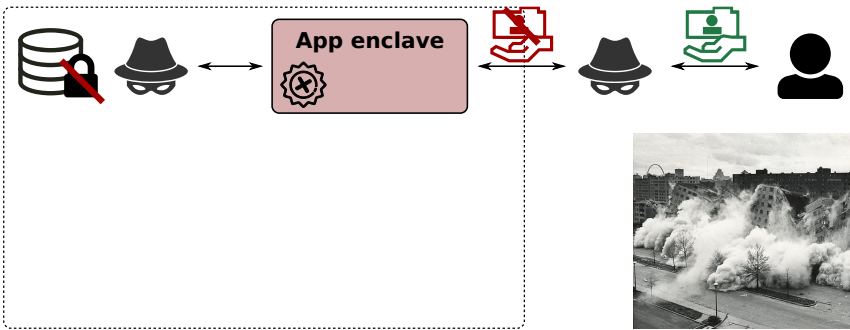
Extract long-term platform **attestation key** → forge Intel signatures



Eroding trust: ~~Remote attestation and secret provisioning~~

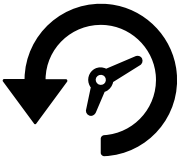
Foreshadow domino effects

Active **man-in-the-middle:** read + modify *all* local and remote secrets (!)



Road map

Foreshadow-NG: Breaking the virtual memory abstraction



1. Cache secrets in L1

2. Unmap page table entry

3. Execute Meltdown

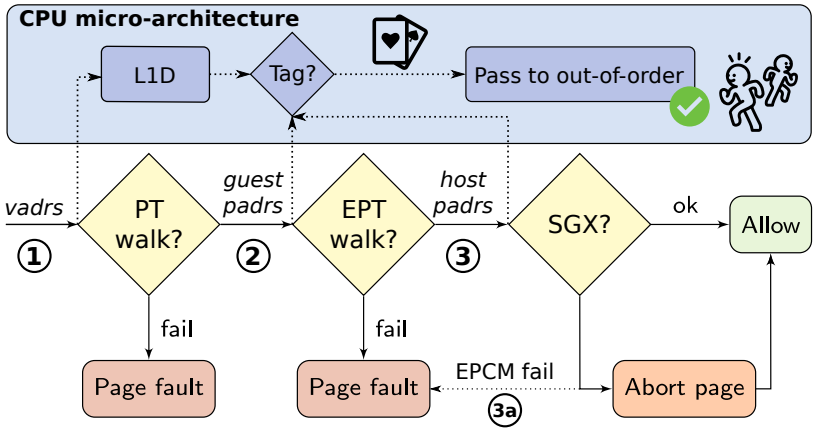
L1 terminal fault [Int18]

Unmap page → read **arbitrary cached physical memory**

https://software.intel.com/security-software-guidance/software-guidance/l1-terminal-fault

Weisse et al. "Foreshadow-NG: Breaking the Virtual Memory Abstraction with Transient Out-of-Order Execution"

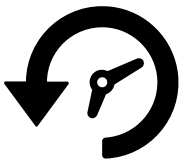
Foreshadow-NG: Breaking the virtual memory abstraction



Weisse et al. "Foreshadow-NG: Breaking the Virtual Memory Abstraction with Transient Out-of-Order Execution"

Road map

Mitigating Foreshadow

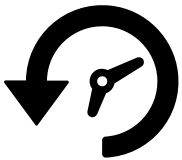


1. Cache secrets in L1

2. Unmap page table entry

3. Execute Meltdown

Mitigating Foreshadow



1. Cache secrets in L1

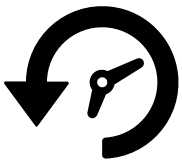
2. Unmap page table entry

3. Execute Meltdown

Future CPUs
(silicon-based changes)

https://newsroom.intel.com/editorials/advancing-security-silicon-level/

Mitigating Foreshadow



1. Cache secrets in L1

2. Unmap page table entry

3. Execute Meltdown

OS kernel updates
(sanitize page frame bits)

Intel SGX: untrusted OS → no software-only mitigations

Mitigating Foreshadow

1. Cache secrets in L1

2. Unmap page table entry

3. Execute Meltdown

Intel microcode updates

⇒ **Flush L1** cache on enclave/VMM exit + **disable HyperThreading**

https://software.intel.com/security-software-guidance/software-guidance/l1-terminal-fault

Conclusions and lessons learned

Take-away message

Foreshadow $==$ *L1 cache read primitive* $\rightarrow$ collapse CPU protection

Conclusions and lessons learned

Take-away message

Foreshadow $==$ *L1 cache read primitive* $\rightarrow$ collapse CPU protection

$\leftrightarrow$ Intel μ-code patches for **TCB recovery** ($+$ disable HyperThreading!)

Conclusions and lessons learned

Take-away message

Foreshadow == *L1 cache read primitive* → collapse CPU protection

↔ Intel μ-code patches for **TCB recovery** (+ disable HyperThreading!)

⇒ Importance of fundamental **side-channel research** (e.g., page table attack surface)

⇒ TEE design: avoid **single point of failure** (domino effects)

Thank you! Questions?

https://foreshadowattack.eu

References I

G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai.
Sgxpectre attacks: Leaking enclave secrets via speculative execution.
arXiv preprint arXiv:1802.09085, 2018.

Intel Corporation.
Intel analysis of L1 terminal fault, August 2018.
https://software.intel.com/security-software-guidance/insights/deep-dive-intel-analysis-l1-terminal-fault.

M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg.
Meltdown: Reading kernel memory from user space.
In *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

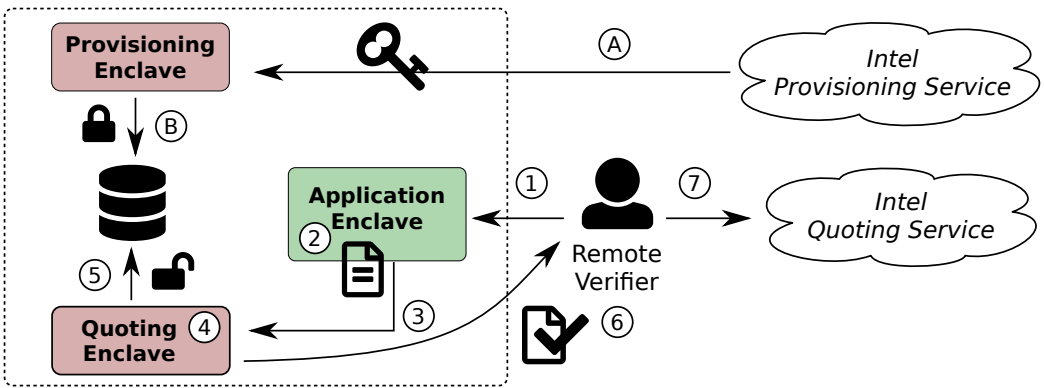
J. Van Bulck, F. Piessens, and R. Strackx.
SGX-Step: A practical attack framework for precise enclave execution control.
In *Proceedings of the 2nd Workshop on System Software for Trusted Execution*, SysTEX'17, pp. 4:1–4:6. ACM, 2017.

J. Van Bulck, N. Weichbrodt, R. Kapitza, F. Piessens, and R. Strackx.
Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution.
In *Proceedings of the 26th USENIX Security Symposium*. USENIX Association, August 2017.

O. Weisse, J. Van Bulck, M. Minkin, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, R. Strackx, T. F. Wenisch, and Y. Yarom.
Foreshadow-NG: Breaking the virtual memory abstraction with transient out-of-order execution.
Technical Report, 2018.

Y. Xu, W. Cui, and M. Peinado.
Controlled-channel attacks: Deterministic side channels for untrusted operating systems.
In *36th IEEE Symposium on Security and Privacy*. IEEE, May 2015.

Appendix: Remote attestation



Appendix: Key derivation