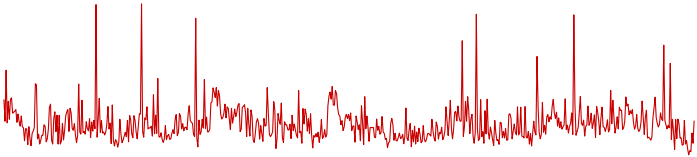
Tutorial: Uncovering Side-Channels in Intel SGX Enclaves

Part 1: Reconstructing enclave code and data accesses

Jo Van Bulck

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

SPACE 2018, December 15, 2018



About imec-DistriNet enclave research

https://distrinet.cs.kuleuven.be/

- Enclave security **across the system stack:** hardware, compiler, OS, application
- Integrated **attack-defense** perspective and **open-source** prototypes

Foreshadow vulnerability
[VBMW+18]

SGX-Step framework
[VBPS17]

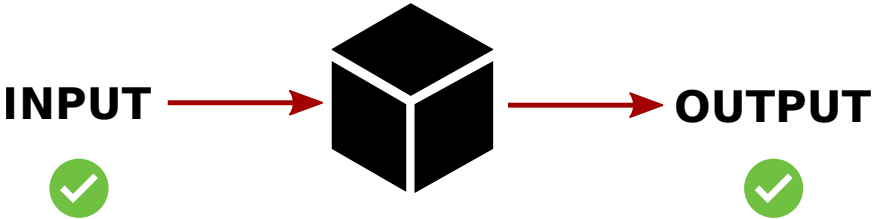
Sancus enclave processor
[NAD+13, NVBM+17]

Tutorial organization

1. **Part 1** (09:00 – 10:30): Reconstructing enclave code and data accesses
   - Lecture: Introduction to Intel SGX and software side-channel attacks
   - Hands-on: Exploiting elementary example applications
2. **Part 2** (11:00 – 12:30): Stealing enclave secrets with transient execution
   - Lecture: Introduction to transient execution attacks (Meltdown, Foreshadow, Spectre)
   - Hands-on: Exploiting elementary example applications

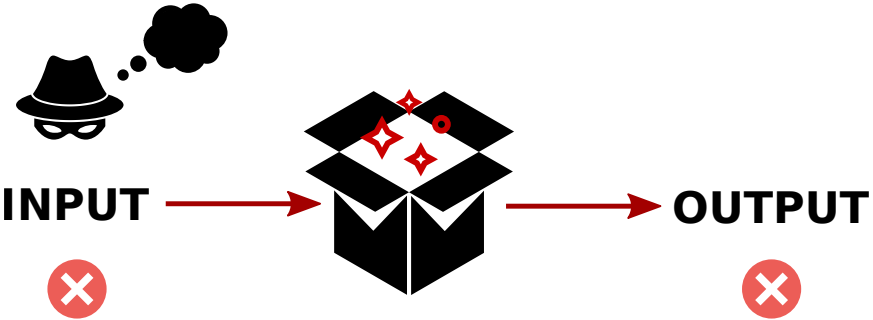
A primer on software security

Secure program: convert all input to *expected output*



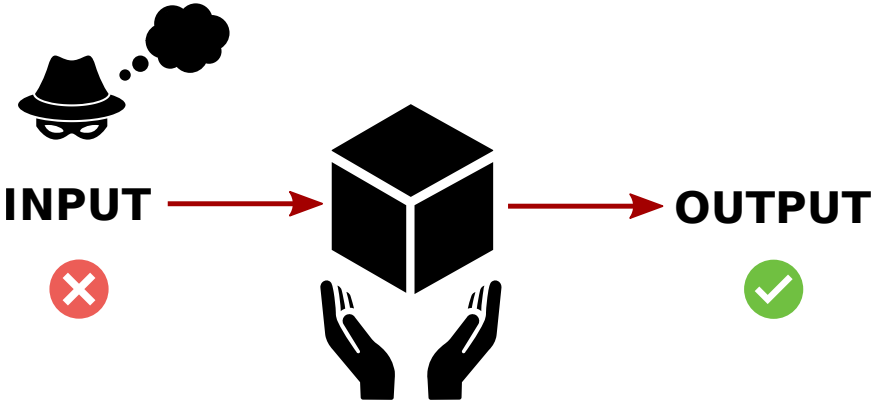
A primer on software security

Buffer overflow vulnerabilities: trigger *unexpected behavior*



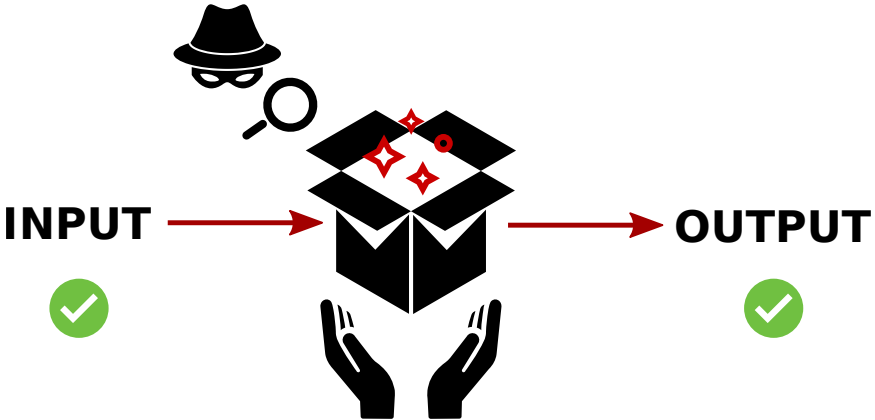
A primer on software security

Safe languages & formal verification: preserve *expected behavior*



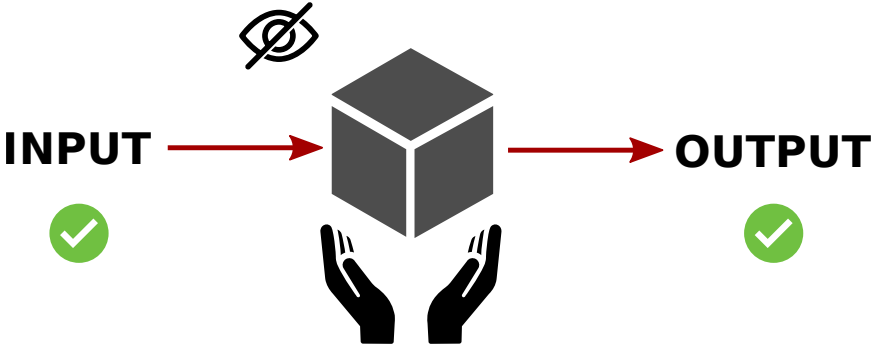
A primer on software security

Side-channels: observe *side-effects* of the computation



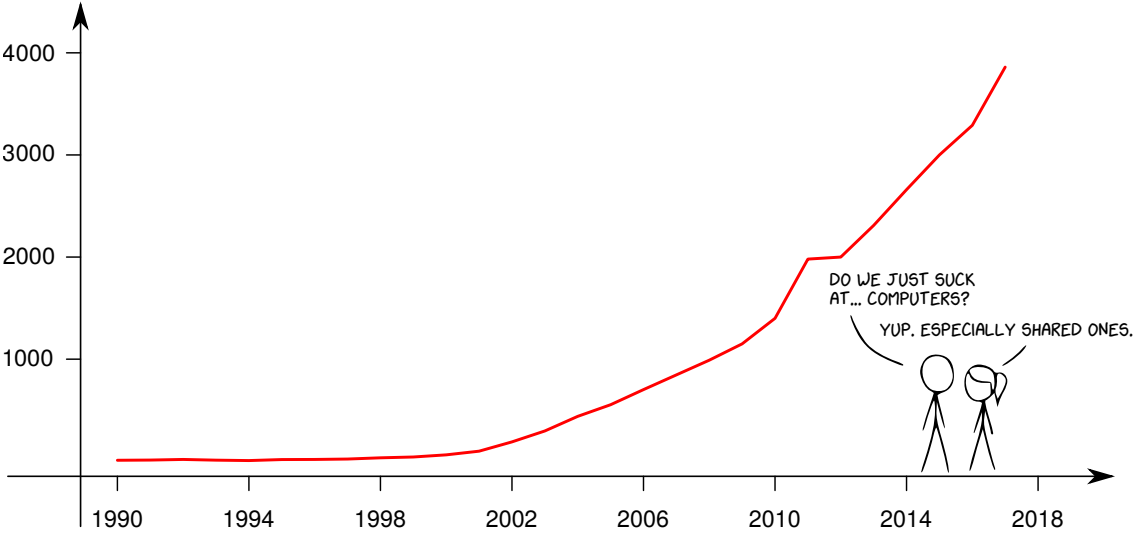
A primer on software security

Constant-time code: eliminate *secret-dependent* side-effects



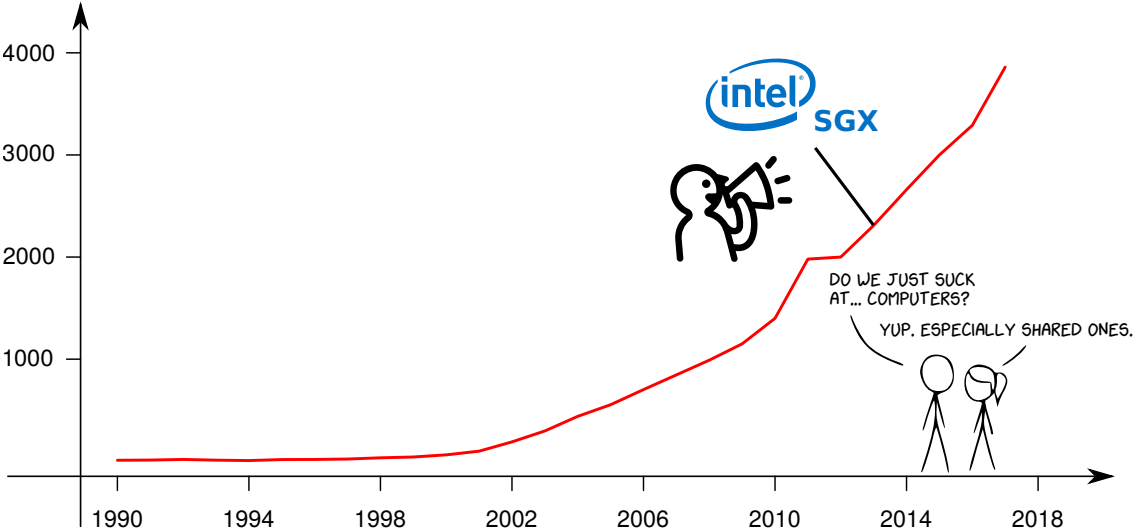

VAULT DOOR

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



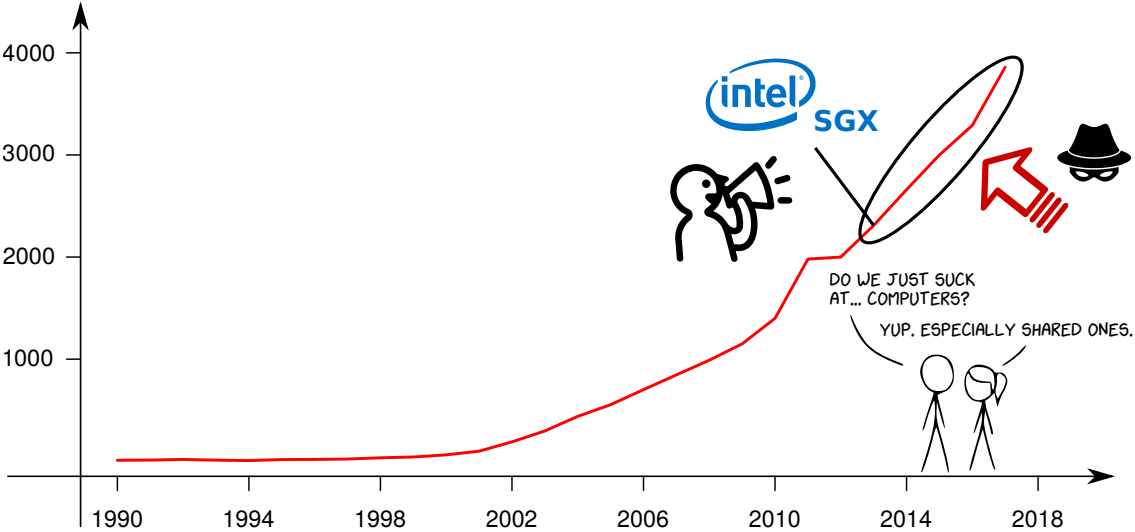
Based on `github.com/Pold87/academic-keyword-occurrence` and `xkcd.com/1938/`

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

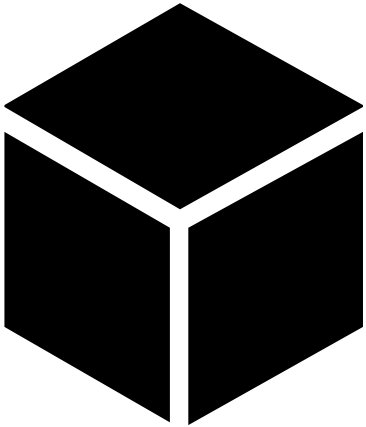


Based on github.com/Pold87/academic-keyword-occurrence and xkcd.com/1938/

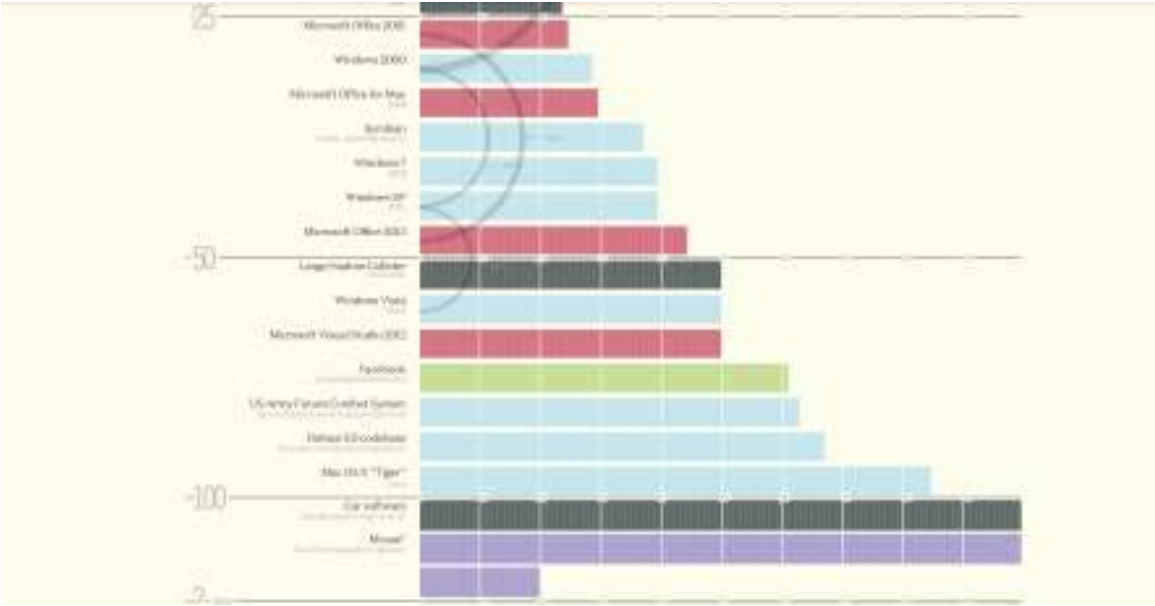
Evolution of “side-channel attack” occurrences in Google Scholar



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

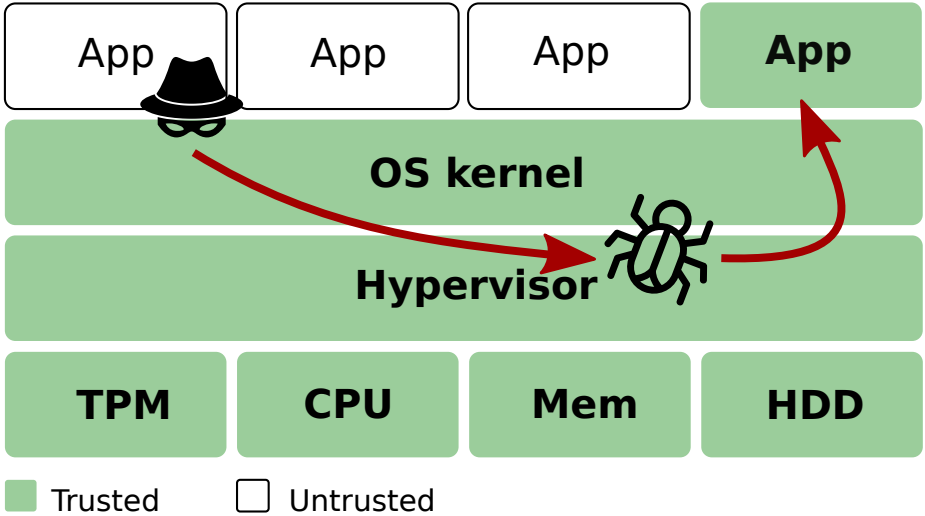


What's inside the black box?

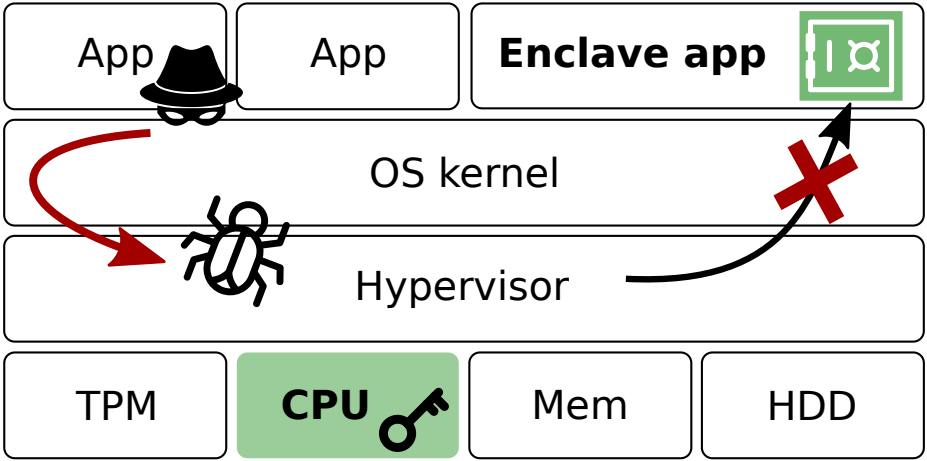


https://informationisbeautiful.net/visualizations/million-lines-of-code/

Enclaved execution: Reducing attack surface

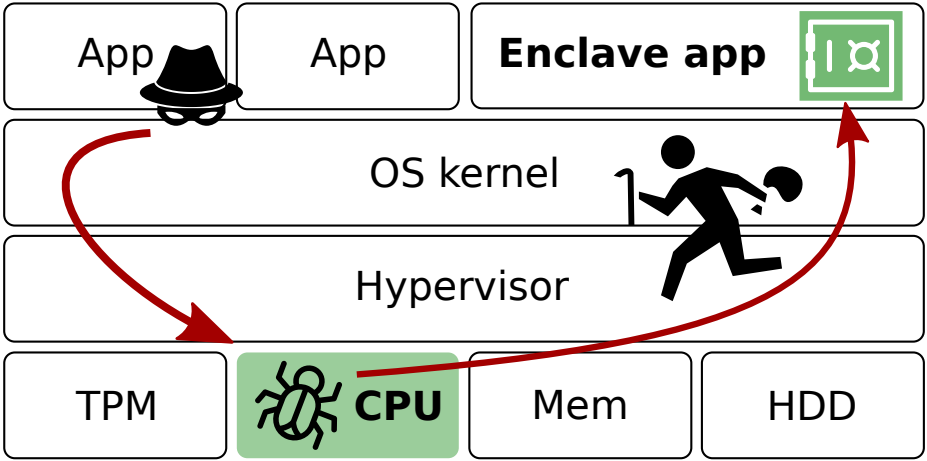


Enclaved execution: Reducing attack surface



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

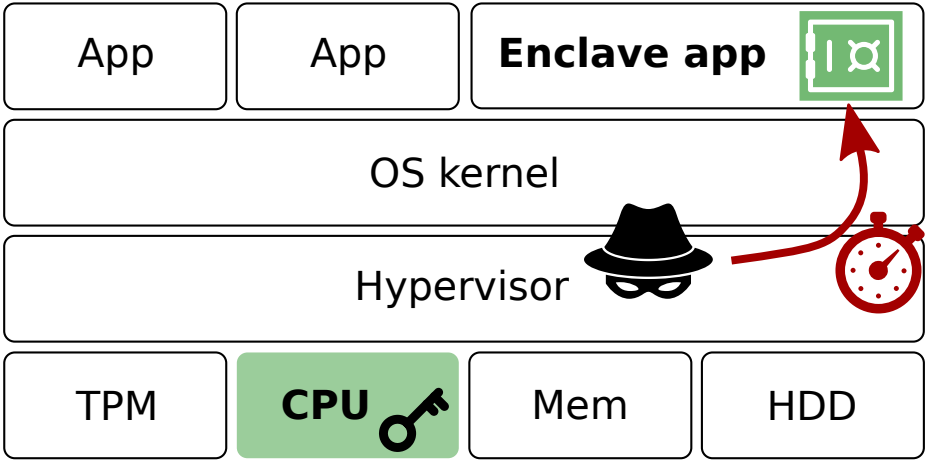
Tutorial part 2: Transient execution attacks



Trusted CPU → exploit **microarchitectural bugs/design flaws**

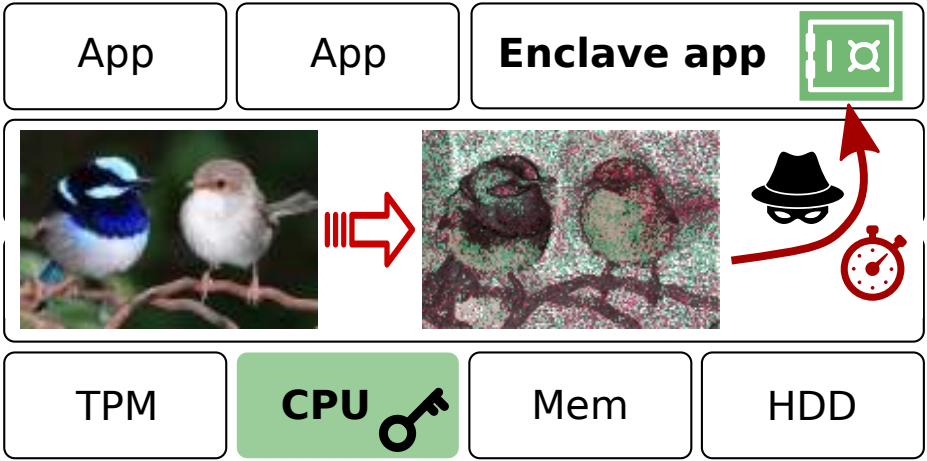
Van Bulck et al. "Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution", USENIX 2018 [VBMW+18]

Tutorial part 1: Privileged side-channel attacks



Untrusted OS → new class of powerful **side-channels**

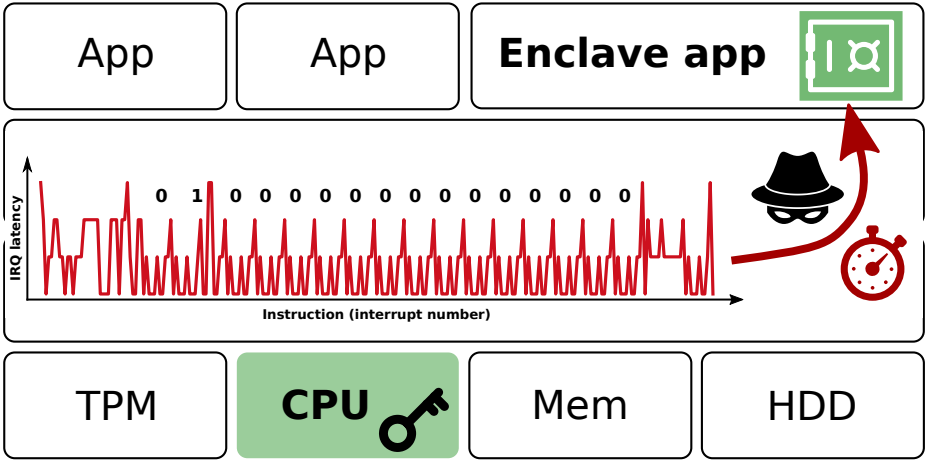
Tutorial part 1: Privileged side-channel attacks



Untrusted OS → new class of powerful **side-channels**

Xu et al. "Controlled-channel attacks: Deterministic side-channels for untrusted operating systems", IEEE S&P 2015 [XCP15]

Tutorial part 1: Privileged side-channel attacks



Untrusted OS → new class of powerful **side-channels**

Van Bulck et al. "Nemesis: Studying Microarchitectural Timing Leaks in Rudimentary CPU Interrupt Logic", CCS 2018 [VBPS18]

KEEP CALM

IT IS

OUT OF SCOPE

A note on 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.

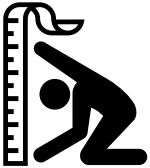
In general, enclave operations that require an OCall, such as thread synchronization, I/O, etc., are exposed to the untrusted domain. If using an OCall would allow an attacker to gain insight into enclave secrets, then there would be a security concern. This scenario would be classified as a side-channel 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.

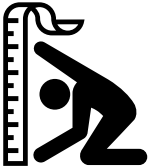
Artwork inspired by Daniel Genkin

Research landscape: Understanding side-channel leakage in enclaves



- Which **side-channels** exist?
- Which enclave **applications** are vulnerable? (Not only crypto!)
- How can we **defend** against them, and at what cost?

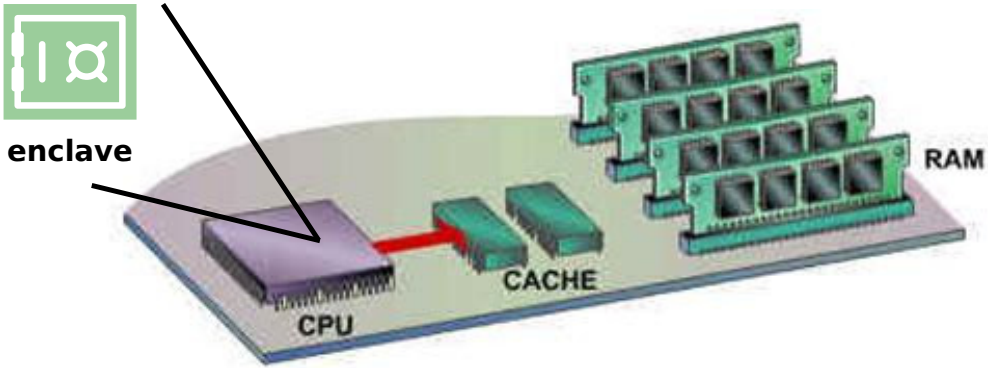
Research landscape: Understanding side-channel leakage in enclaves



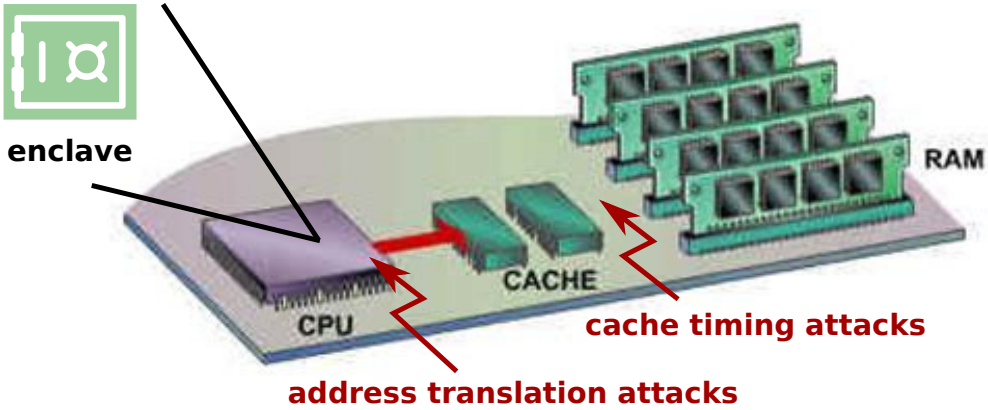
- Which **side-channels** exist?
- Which enclave **applications** are vulnerable? (Not only crypto!)
- How can we **defend** against them, and at what cost?

⇒ Educate developers to raise awareness and avoid side-channel pitfalls (= this tutorial!)

Overview: Spying on enclave memory accesses



Overview: Spying on enclave memory accesses



Secret-dependent code/data memory accesses

```c
void secret_vote(char candidate)
{
    if (candidate == 'a')
        vote_candidate_a();
    else
        vote_candidate_b();
}
```

```c
int secret_lookup(int s)
{
    if (s > 0 && s < ARRAY_LEN)
        return array[s];
    return -1;

}
```

Secret-dependent code/data memory accesses

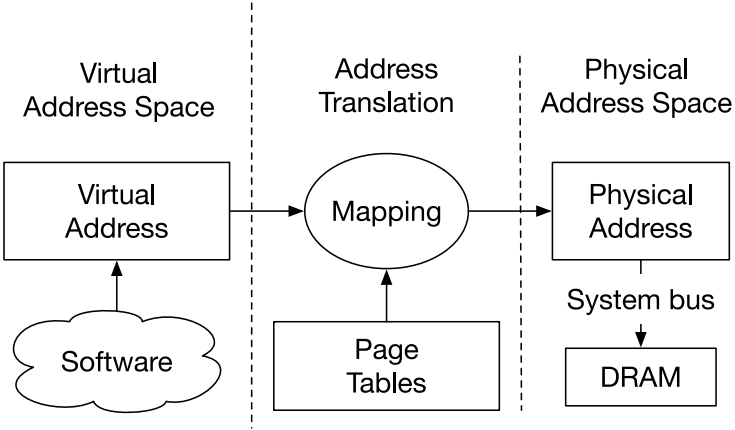
```c
void secret_vote(char candidate)
{
    if (candidate == 'a')
        vote_candidate_a();
    else
        vote_candidate_b();
}
```

```c
int secret_lookup(int s)
{
    if (s > 0 && s < ARRAY_LEN)
        return array[s];
    return -1;

}
```

What if the adversary obtains a perfect "oracle" for all enclaved code+data memory access sequences?

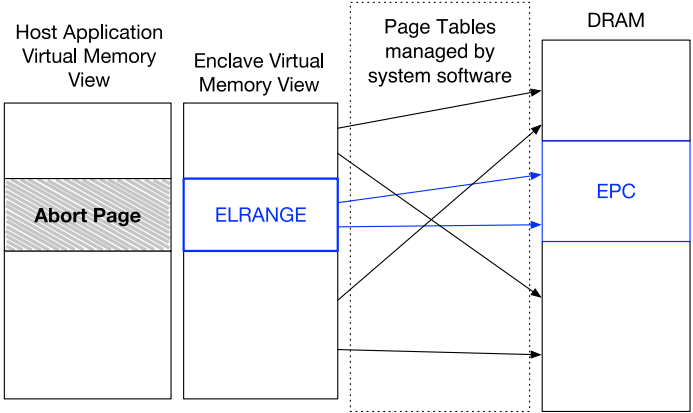
Address translation attacks

The virtual memory abstraction



Costan et al. "Intel SGX explained", IACR 2016 [CD16]

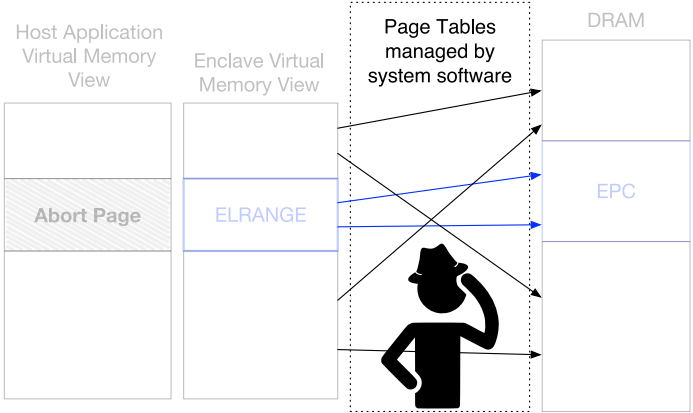
How enclave accesses are enforced



Costan et al. "Intel SGX explained", IACR 2016 [CD16]

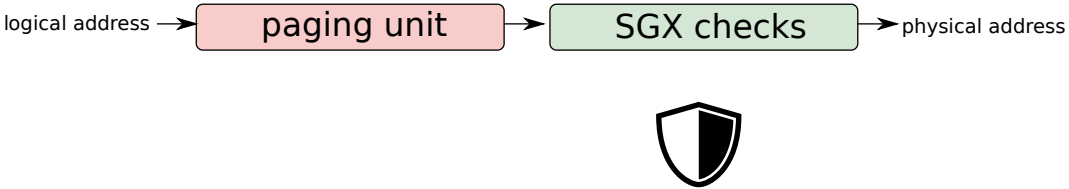
How enclave accesses are enforced

Note: Untrusted OS controls *virtual-to-physical mapping*



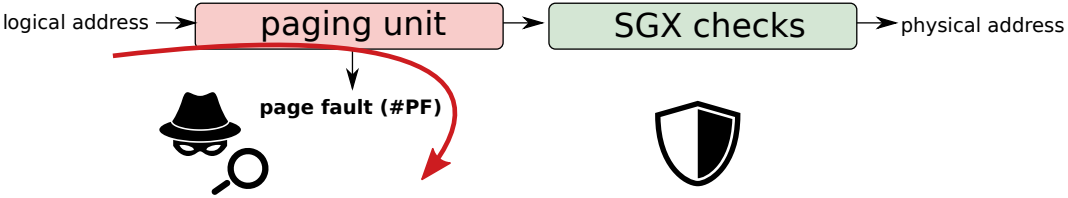
Costan et al. "Intel SGX explained", IACR 2016 [CD16]

Page faults as a side-channel



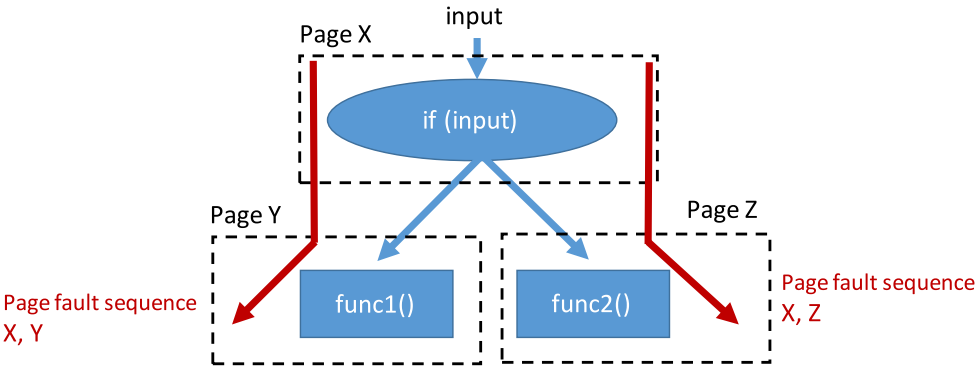
SGX machinery protects against direct address remapping attacks

Page faults as a side-channel



. . . but untrusted address translation may **fault** during enclaved execution (!)

Page faults as a side-channel



Xu et al.: "Controlled-channel attacks: Deterministic side channels for untrusted operating systems", Oakland 2015 [XCP15]

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

#PF attacks: An end-to-end example

```
void inc_secret( void )
{
    if (secret)
        *a += 1;
    else
        *b += 1;
}
```

Page Table

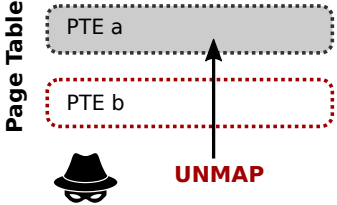
PTE a

PTE b

#PF attacks: An end-to-end example

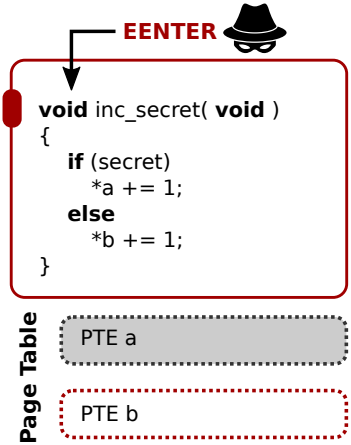
1. Revoke access rights on *unprotected* enclave page table entry

```
void inc_secret( void )
{
    if (secret)
        *a += 1;
    else
        *b += 1;
}
```



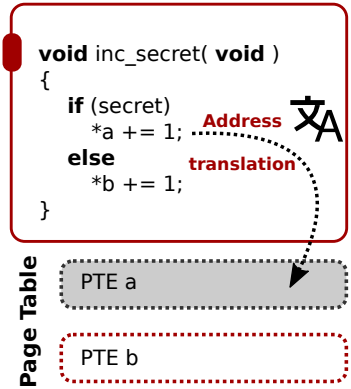
#PF attacks: An end-to-end example

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



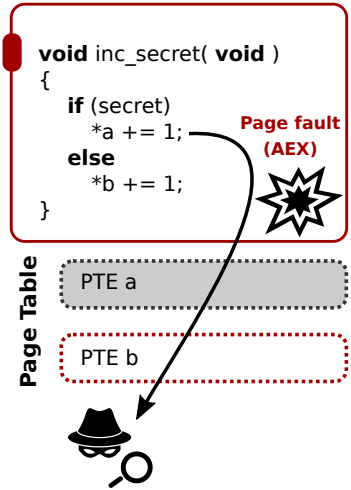
#PF attacks: An end-to-end example

1. Revoke access rights on *unprotected* enclave page table entry
2. Enter victim enclave
3. Secret-dependent data memory access
   - ⇝ Processor performs virt-to-phys address translation!



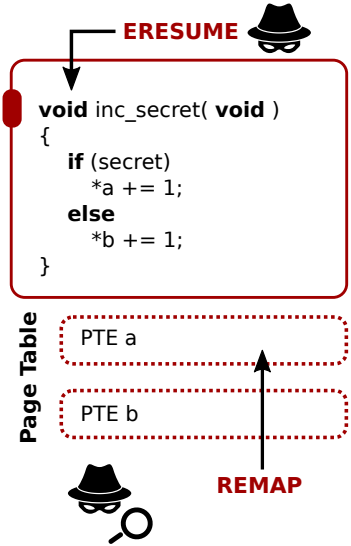
#PF attacks: An end-to-end example

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

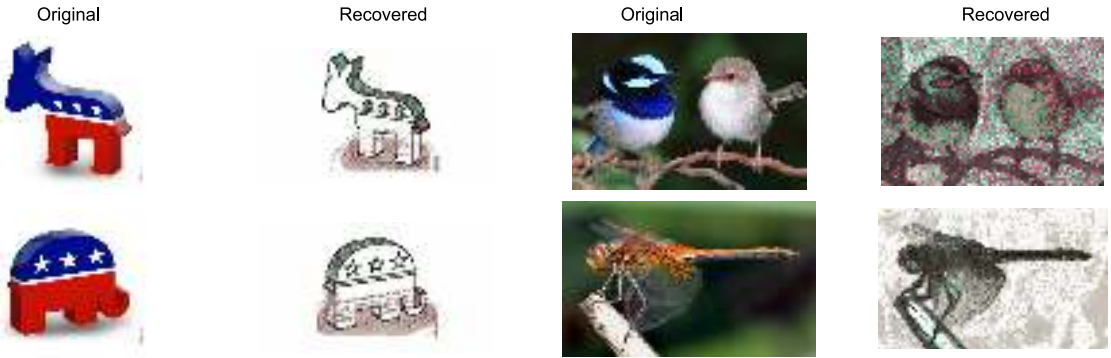


#PF attacks: An end-to-end example

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



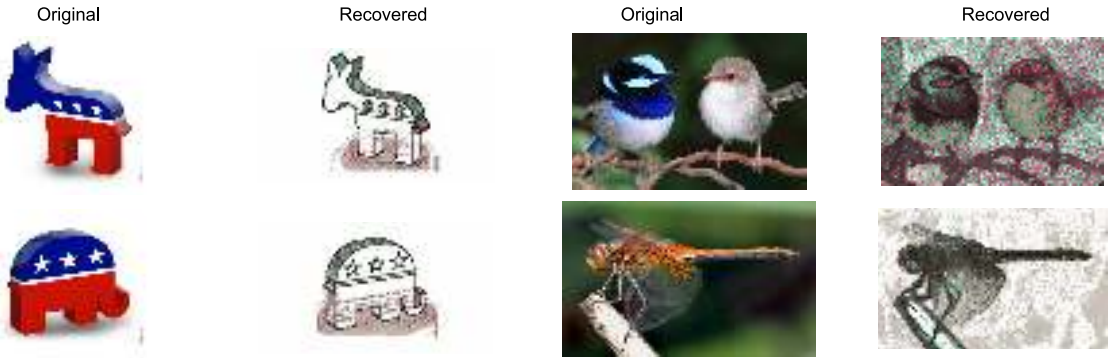
Page table-based attacks in practice



Xu et al.: "Controlled-channel attacks: Deterministic side channels for untrusted operating systems", Oakland 2015 [XCP15]

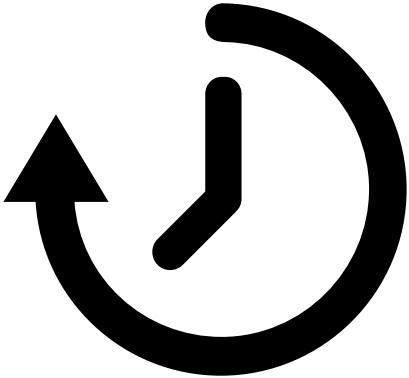
⇒ **Low-noise, single-run** exploitation of legacy applications

Page table-based attacks in practice



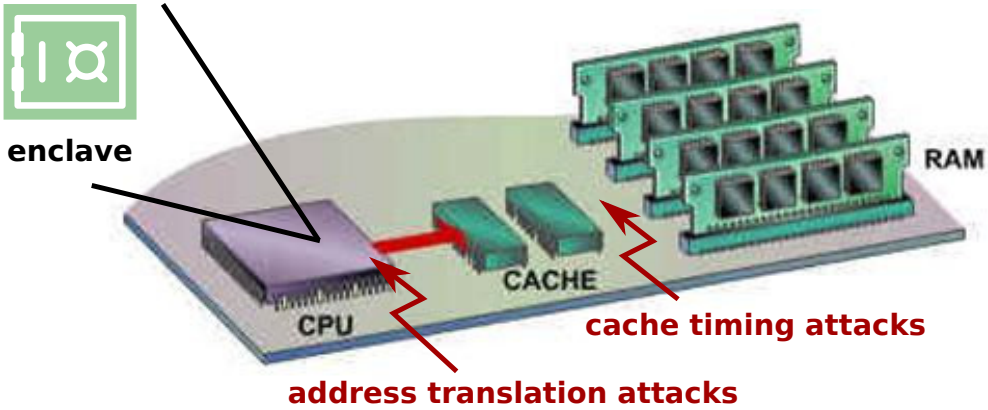
Xu et al.: "Controlled-channel attacks: Deterministic side channels for untrusted operating systems", Oakland 2015 [XCP15]

. . . but at a relative coarse-grained **4 KiB granularity**

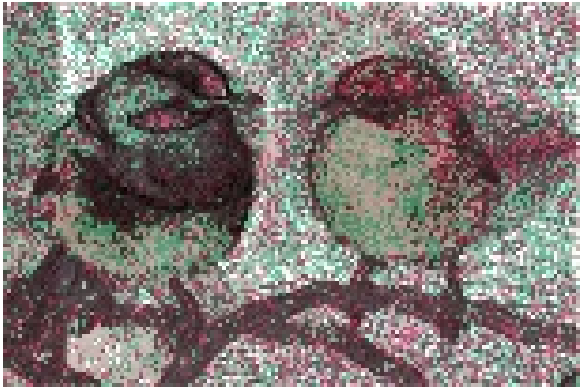


Cache timing attacks

Overview: Spying on enclave code/data accesses (revisited)



High resolution side-channels in practice



Xu et al.: "Controlled-channel attacks: Deterministic side channels for untrusted operating systems", Oakland 2015 [XCP15]

$\Longrightarrow$ Coarse-grained preemption **(4 KB page leakage)**

High resolution side-channels in practice

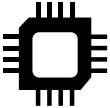
Hähnel et al.: "High-resolution side channels for untrusted operating systems", ATC 2017 [HCP17]

$\Longrightarrow$ Fine-grained preemption **(64 B cache line leakage)**

CPU cache timing side-channel

Cache principle: CPU speed $\gg$ DRAM latency $\rightarrow$ *cache code/data*

```
while true do
    maccess(&a);
endwh
```



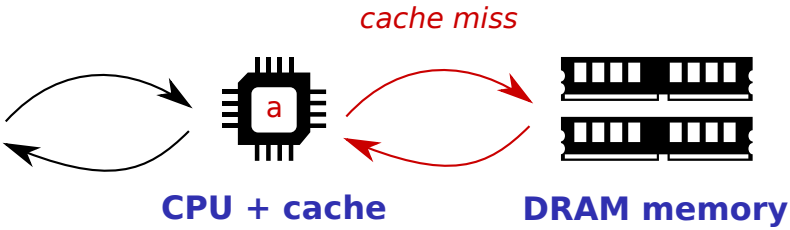
CPU + cache

DRAM memory

CPU cache timing side-channel

Cache miss: Request data from (slow) DRAM upon first use

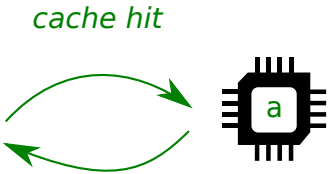
```
while true do
    maccess(&a);
endwh
```



CPU cache timing side-channel

Cache hit: No DRAM access required for subsequent uses

```
while true do
    maccess(&a);
endwh
```



CPU + cache

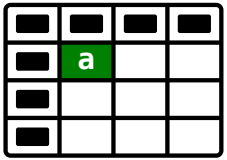
DRAM memory

Flush+Reload: Cache timing attacks on shared memory

```
if secret do
    maccess(&a);
else
    maccess(&b);
endif
```

```
flush(&a);
start_timer
   maccess(&a);
end_timer
```

'a' is accessible to attacker



CPU cache

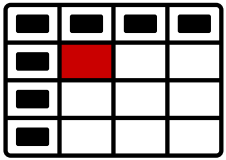
DRAM memory

Flush+Reload: Cache timing attacks on shared memory

```
if secret do
    maccess(&a);
else
    maccess(&b);
endif
```

```
flush(&a);
start_timer
    maccess(&a);
end_timer
```

flush 'a' to memory



CPU cache

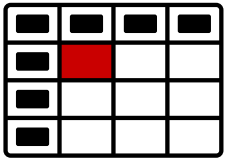
DRAM memory

Flush+Reload: Cache timing attacks on shared memory

```
if secret do
    maccess(&a);
else
    maccess(&b);
endif
```

```
flush(&a);
start_timer
   maccess(&a);
end_timer
```

cache miss



CPU cache

DRAM memory

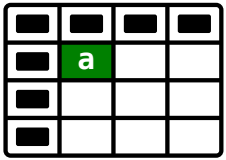
Flush+Reload: Cache timing attacks on shared memory

```
if secret do
    maccess(&a);
else
    maccess(&b);
endif
```

```
flush(&a);
start_timer
   maccess(&a);
end_timer
```

cache miss

secret=1, load 'a' from memory



CPU cache

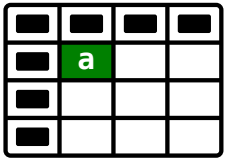
DRAM memory

Flush+Reload: Cache timing attacks on shared memory

```
if secret do
    maccess(&a);
else
    maccess(&b);
endif
```

```
flush(&a);
start_timer
    maccess(&a);
end_timer
```

cache hit



CPU cache

DRAM memory

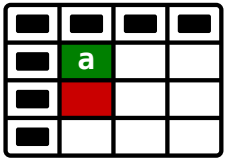
fast access(&a) → secret=1

Flush+Reload: Cache timing attacks on shared memory

```
if secret do
    maccess(&a);
else
    maccess(&b);
endif
```

```
flush(&a);
start_timer
    maccess(&b);
end_timer
```

cache miss



CPU cache

DRAM memory

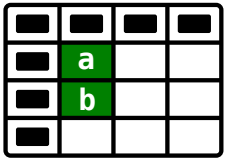
Flush+Reload: Cache timing attacks on shared memory

```
if secret do
    maccess(&a);
else
    maccess(&b);
endif
```

```
flush(&a);
start_timer
    maccess(&b);
end_timer
```

load 'b' from memory

cache miss

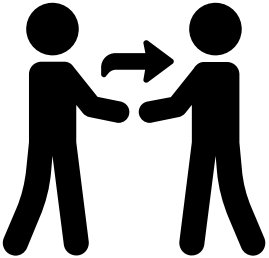


CPU cache

DRAM memory

slow access(&b) → secret=0

Flush+Reload limitations

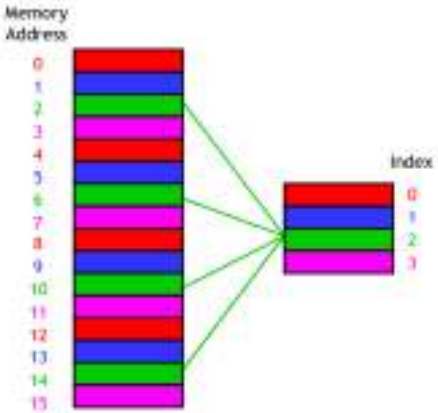


- Very **reliable** attack + easy to mount
- . . . but relies on **shared memory** ($\leftrightarrow$ enclaves)!


prime day

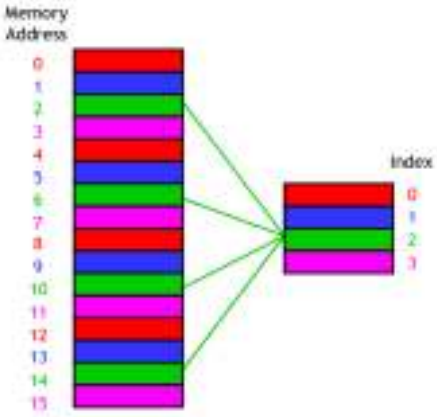
CPU cache organization 101

- **Shared** among all protection domains ☺
- Cache size ≪ addressable memory size
- **Cache line:** unit of caching (64 bytes)
- **Mapping scheme:** memory address → cache line



CPU cache organization 101

- **Shared** among all protection domains ☺
- Cache size ≪ addressable memory size
- **Cache line:** unit of caching (64 bytes)
- **Mapping scheme:** memory address → cache line
- **Cache collision:** replace cache line with new data requested from memory

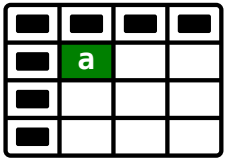


Prime+Probe: Cache timing attacks across protection domains

```
if secret do
    maccess(&a);
else
    maccess(&b);
endif
```

```
maccess(&c);
start_timer
   maccess(&c);
end_timer
```

*'a' is **not** accessible to attacker*



CPU cache

DRAM memory

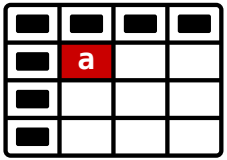
Prime+Probe: Cache timing attacks across protection domains

```
if secret do
    maccess(&a);
else
    maccess(&b);
endif
```

```
maccess(&c);
start_timer
   maccess(&c);
end_timer
```

cache miss

'a' and 'c' share the same cache line (!)



CPU cache

DRAM memory

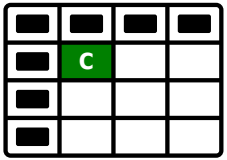
Prime+Probe: Cache timing attacks across protection domains

```
if secret do
    maccess(&a);
else
    maccess(&b);
endif
```

```
maccess(&c);
start_timer
   maccess(&c);
end_timer
```

load 'c' from memory (write back 'a')

cache miss



CPU cache

DRAM memory

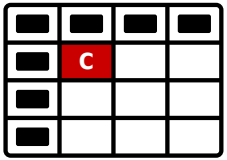
Prime+Probe: Cache timing attacks across protection domains

```
if secret do
    maccess(&a);
else
    maccess(&b);
endif
```

```
flush(&c);
start_timer
    maccess(&c);
end_timer
```

cache miss

'a' and 'c' share the same cache line (!)



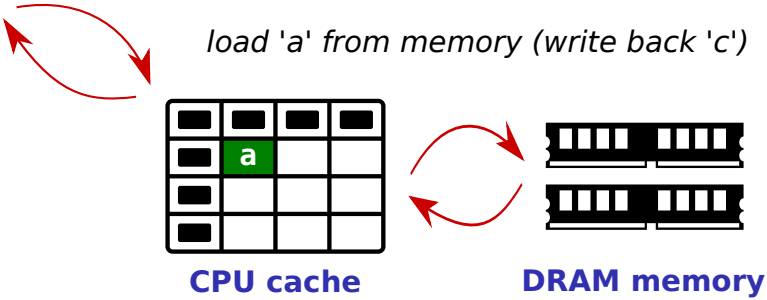
CPU cache

DRAM memory

Prime+Probe: Cache timing attacks across protection domains

```
if secret do
    maccess(&a);
else
    maccess(&b);
endif
```

```
flush(&c);
start_timer
    maccess(&c);
end_timer
```



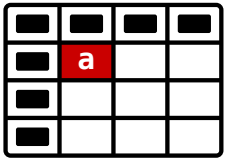
Prime+Probe: Cache timing attacks across protection domains

```
if secret do
    maccess(&a);
else
    maccess(&b);
endif
```

```
flush(&c);
start_timer
    maccess(&c);
end_timer
```

cache miss

'a' and 'c' share the same cache line (!)

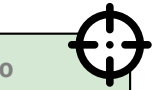


CPU cache

DRAM memory

Prime+Probe: Cache timing attacks across protection domains

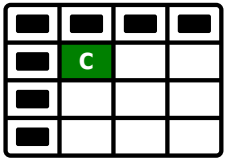
```
if secret do
    maccess(&a);
else
    maccess(&b);
endif
```



```
flush(&c);
start_timer
    maccess(&c);
end_timer
```

load 'c' from memory (write back 'a')

cache miss

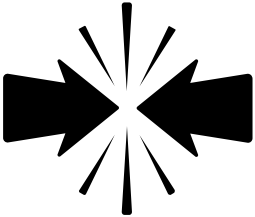


CPU cache

DRAM memory

slow access(&c) → secret=1

Prime+Probe Challenges

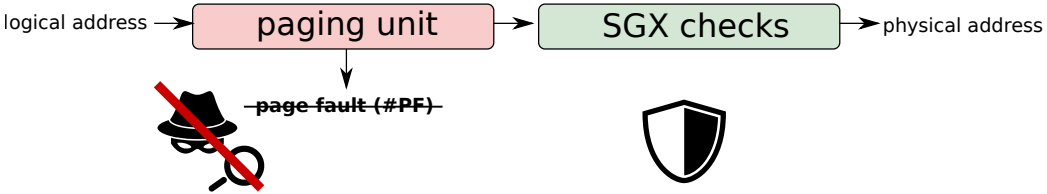


- Exploit **contention** on shared cache resource
- Very **generic** attack applicable to many cache designs + protection domains
- . . . but relies on detailed understanding of **cache mapping** scheme → complex for real-world set-associative caches (e.g., reverse engineering Intel last-level cache)

Defense idea #1

What about hiding enclave page faults?

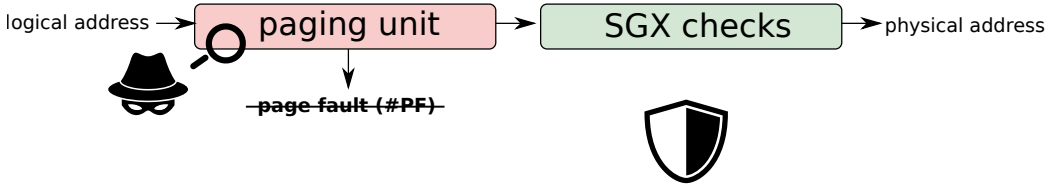
Current solutions: Hiding enclave page faults



Shih et al. "T-SGX: Eradicating controlled-channel attacks against enclave programs", NDSS 2017 [SLKP17]

Shinde et al. "Preventing page faults from telling your secrets", AsiaCCS 2016 [SCNS16]

Current solutions: Hiding enclave page faults

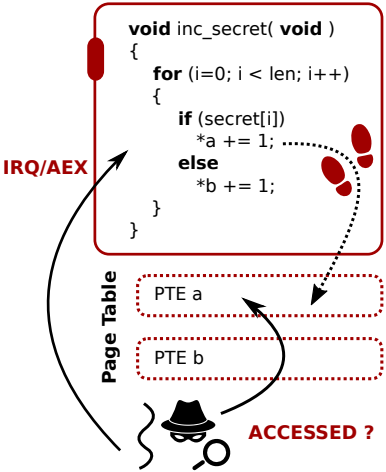


. . . But stealthy attacker can still learn page accesses without triggering faults!

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 [VBWK+17]

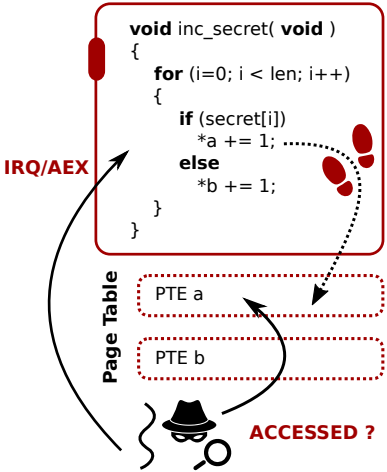
Telling your secrets without page faults

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

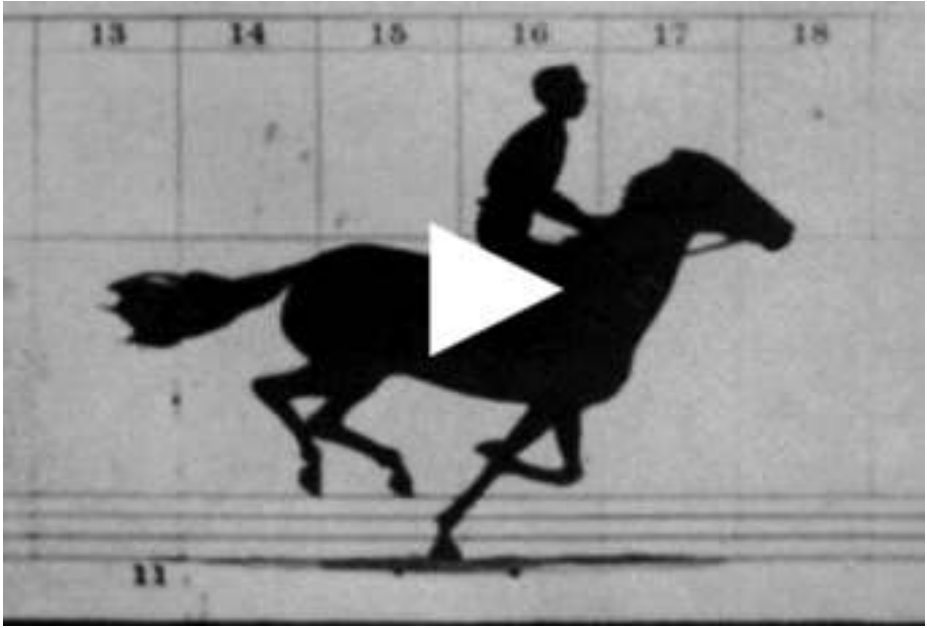
   ⇝ Flush+Reload cache timing attack!



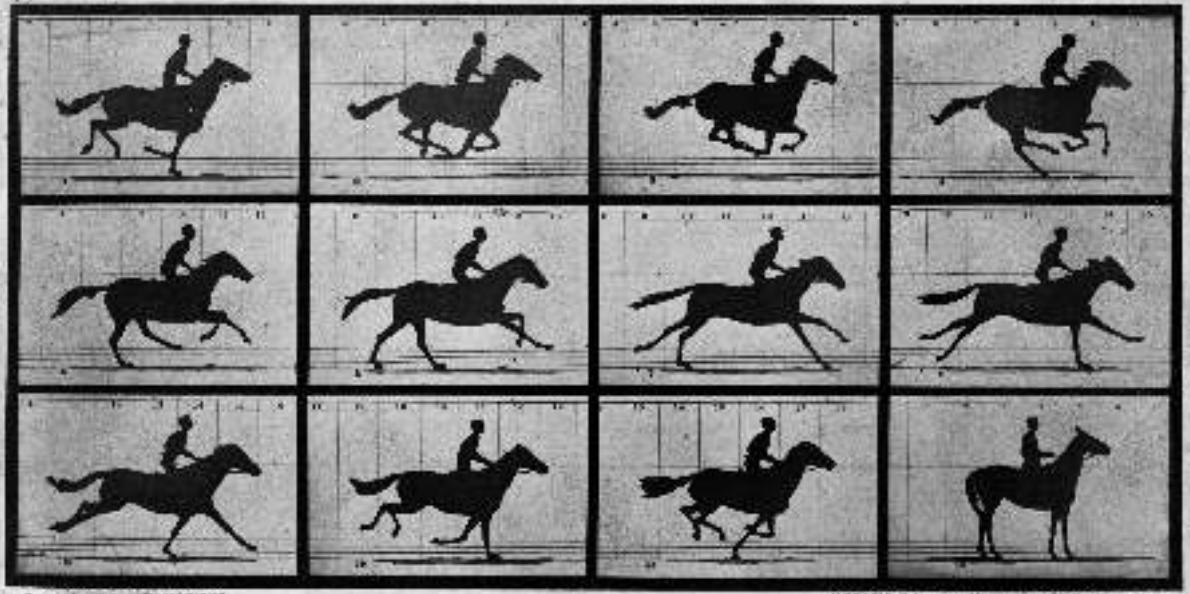
Van Bulck et al. "Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution", USENIX 2017 [VBWK+17]

Defense idea #2

What about limiting the temporal resolution?



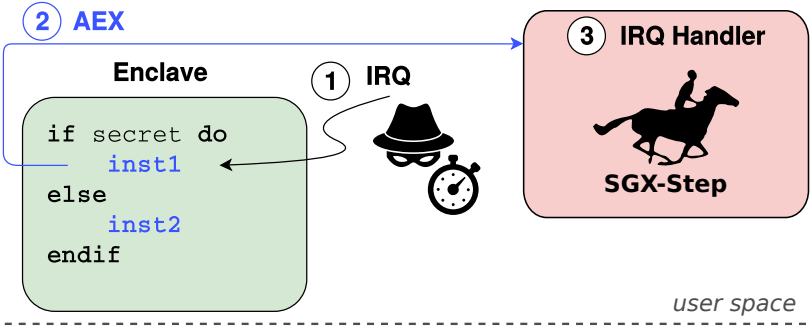
https://en.wikipedia.org/wiki/Sallie_Gardner_at_a_Gallop



THE HORSE IN MOTION.

SGX-Step: Executing enclaves one instruction at a time

SGX-Step: user space APIC timer + interrupt handling ☺



Van Bulck et al. "SGX-Step: A practical attack framework for precise enclave execution control", SysTEX 2017 [VBPS17]

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

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

Protection from Side-Channel Attacks

Intel® SGX does not provide explicit protection from side-channel attacks. It is the enclave developer's responsibility to address side-channel attack concerns.

In general, enclave operations that require an OCall, such as thread synchronization, I/O, etc., are exposed to the untrusted domain. If using an OCall would allow an attacker to gain insight into enclave secrets, then there would be a security concern. This scenario would be classified as a side-channel attack, and it would be up to the ISV to design the enclave in a way that prevents the leaking of side-channel information.

An attacker with access to the platform can see what pages are being executed or accessed. This side-channel vulnerability can be mitigated by aligning specific code and data blocks to exist entirely within a single page.

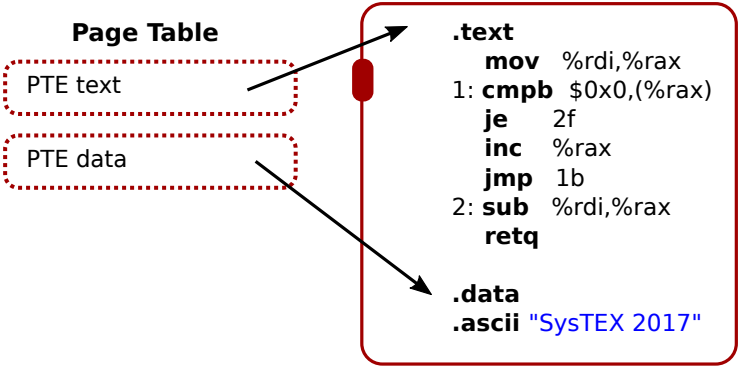
More important, the application enclave should use an appropriate crypto implementation that is side channel attack resistant inside the enclave if side-channel attacks are a concern.

https://software.intel.com/en-us/node/703016

High-resolution attacks in practice: Attacking `strlen`

Page fault adversary

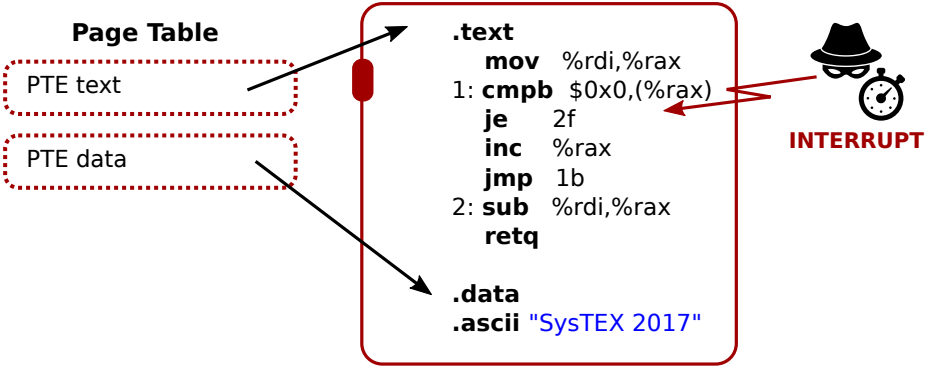
Progress ⇒ both code + data pages present ☹



High-resolution attacks in practice: Attacking `strlen`

Single-stepping adversary

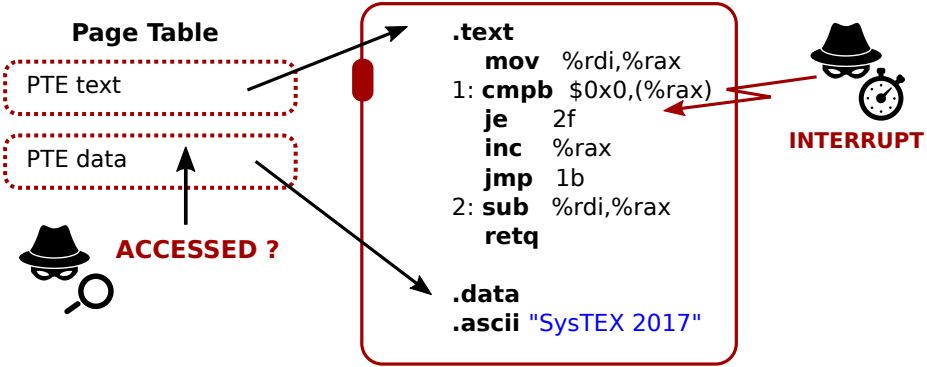
Execute + interrupt ⟹ data page accessed ? ☺



High-resolution attacks in practice: Attacking `strlen`

Single-stepping adversary

Execute + interrupt ⇒ data page accessed ? ☺




Theory
Into
Practice

Hands-on exercises

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

Important note

First develop the *unprotected attack scenario on your local x86 machine*, before testing the enclaved version on the remote SGX machine via SSH (!)

1. Connect to the `space18-sgx` **WiFi network**
   - WPA2 passphrase "space2018-sgx-tutorial"
2. Now ssh into the **SGX machine:** `ssh sgx@10.45.160.95`
   - User: "sgx"
   - Password: "space18"
   - Make sure to work in your own directory to avoid interference

References I

V. Costan and S. Devadas.
Intel SGX explained.
Cryptology ePrint Archive, Report 2016/086, 2016.

M. Hähnel, W. Cui, and M. Peinado.
High-resolution side channels for untrusted operating systems.
In *2017 USENIX Annual Technical Conference*, ATC '17. USENIX Association, 2017.

J. Noorman, P. Agten, W. Daniels, R. Strackx, A. Van Herrewege, C. Huygens, B. Preneel, I. Verbauwhede, and F. Piessens.
Sancus: Low-cost trustworthy extensible networked devices with a zero-software trusted computing base.
In *22nd USENIX Security symposium*, pp. 479–494. USENIX Association, 2013.

J. Noorman, J. Van Bulck, J. T. Mühlberg, F. Piessens, P. Maene, B. Preneel, I. Verbauwhede, J. Götzfried, T. Müller, and F. Freiling.
Sancus 2.0: A low-cost security architecture for IoT devices.
ACM Transactions on Privacy and Security (TOPS), 2017.

S. Shinde, Z. L. Chua, V. Narayanan, and P. Saxena.
Preventing page faults from telling your secrets.
In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security (ASIA CCS)*, pp. 317–328. ACM, 2016.

M.-W. Shih, S. Lee, T. Kim, and M. Peinado.
T-SGX: Eradicating controlled-channel attacks against enclave programs.
In *Proceedings of the 2017 Network and Distributed System Security Symposium (NDSS 2017)*, February 2017.

J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx.
Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution.
In *Proceedings of the 27th USENIX Security Symposium*. USENIX Association, August 2018.

References II

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, F. Piessens, and R. Strackx.
Nemesis: Studying microarchitectural timing leaks in rudimentary CPU interrupt logic.
In *Proceedings of the 25th ACM Conference on Computer and Communications Security (CCS'18)*. ACM, October 2018.

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.

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.

Tutorial: Uncovering Side-Channels in Intel SGX Enclaves

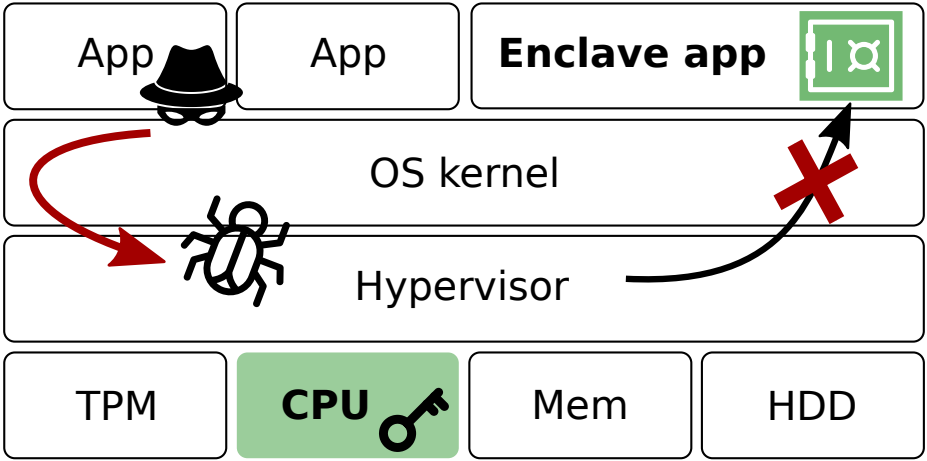
Part 2: Stealing enclave secrets with transient execution

Jo Van Bulck

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

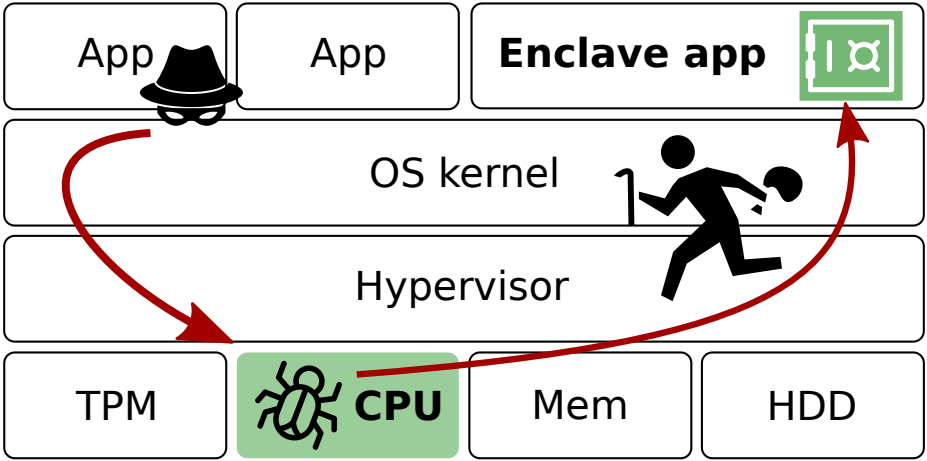
SPACE 2018, December 15, 2018

Enclaved execution attack surface (revisited)



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

Enclaved execution attack surface (revisited)



Trusted CPU → exploit **microarchitectural bugs/design flaws**

Van Bulck et al. "Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution", USENIX 2018 [VBMW+18]

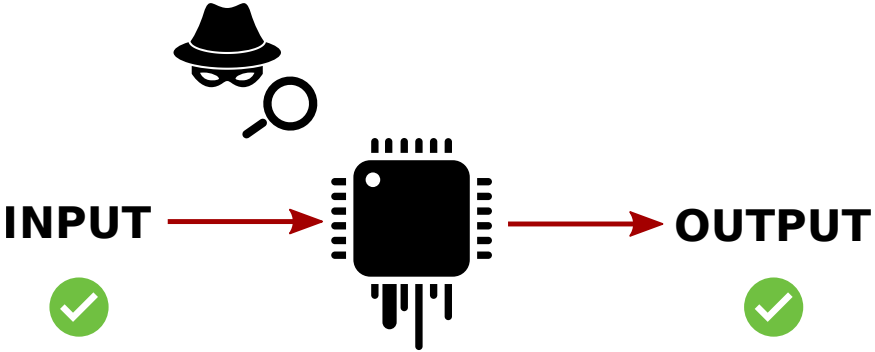
Reflections on trusting trust

"No amount of source-level verification or scrutiny will protect you from using untrusted code. [...] As the level of program gets lower, these bugs will be harder and harder to detect. A well installed microcode bug will be almost impossible to detect."

— Ken Thompson (ACM Turing award lecture, 1984)

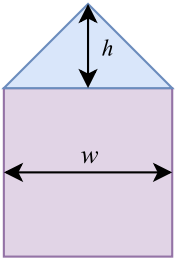
A primer on software security (revisited)

Transient execution: *HW optimizations* do not respect SW abstractions (!)




WHAT IF I TOLD YOU
YOU CAN CHANGE RULES MID-GAME

Out-of-order and speculative execution

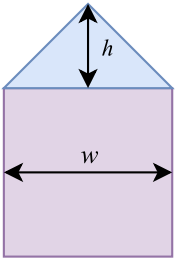



```
int area(int h, int w)
{
  int triangle = (w*h)/2;
  int square   = (w*w);
  return triangle + square;
}
```

Key **discrepancy:**

- Programmers write sequential instructions

Out-of-order and speculative execution



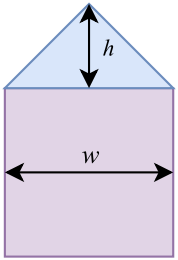

```
int area(int h, int w)
{
  int triangle = (w*h)/2;
  int square   = (w*w);
  return triangle + square;
}
```

Key **discrepancy:**

- Programmers write sequential instructions
- Modern CPUs are inherently parallel

⇒ *Speculatively execute instructions ahead of time*

Out-of-order and speculative execution



Roll-back

Overflow exception

```
int area(int h, int w)
{
  int triangle = (w*h)/2;
  int square   = (w*w);
  return triangle + square;
}
```

Key **discrepancy:**

- Programmers write sequential instructions
- Modern CPUs are inherently parallel

⇒ *Speculatively execute instructions ahead of time*

Best-effort: What if triangle fails?

→ Commit in-order, roll-back square

... But side-channels may leave traces (!)


STRANGER THINGS
EXPLORING THE UPSIDE DOWN

Transient execution attacks: Welcome to the world of fun!

CPU executes ahead of time in **transient world**

- Success → *commit* results to normal world ☺
- Fail → *discard* results, compute again in normal world ☹

Transient execution attacks: Welcome to the world of fun!

CPU executes ahead of time in **transient world**

- Success → *commit* results to normal world ☺
- Fail → *discard* results, compute again in normal world ☹

Transient world (microarchitecture) may temp bypass <u>architectural software intentions:</u>

Delayed exception handling

Control flow prediction

Transient execution attacks: Welcome to the world of fun!

Key finding of 2018

$\Rightarrow$ *Transmit secrets from transient to normal world*

Transient world (microarchitecture) may temp bypass <u>architectural software intentions:</u>

Delayed exception handling

Control flow prediction

Transient execution attacks: Welcome to the world of fun!

Key finding of 2018

$\Rightarrow$ *Transmit secrets from transient to normal world*

Transient world (microarchitecture) may temp bypass <u>architectural software intentions:</u>

CPU access control bypass

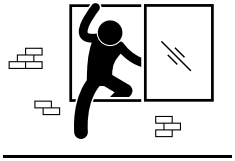
Speculative buffer overflow/ROP


inside™


inside™


inside™

Meltdown: Transiently encoding unauthorized memory



Unauthorized access

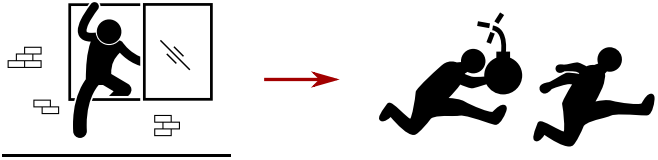
Listing 1: x86 assembly

```
meltdown:
  // %rdi: oracle
  // %rsi: secret_ptr

  movb (%rsi), %al
  shl $0xc, %rax
  movq (%rdi, %rax), %rdi
  retq
```

Listing 2: C code.

```
void meltdown(
        uint8_t *oracle,
        uint8_t *secret_ptr)
{
  uint8_t v = *secret_ptr;
  v = v * 0x1000;
  uint64_t o = oracle[v];
}
```

Meltdown: Transiently encoding unauthorized memory



Unauthorized access

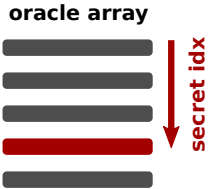
Transient out-of-order window

Listing 1: x86 assembly.

```
meltdown:
  // %rdi: oracle
  // %rsi: secret_ptr

  movb (%rsi), %al
  shl $0xc, %rax
  movq (%rdi, %rax), %rdi
  retq
```

Listing 2: C code.

```
void meltdown(
        uint8_t *oracle,
        uint8_t *secret_ptr)
{
  uint8_t v = *secret_ptr;
  v = v * 0x1000;
  uint64_t o = oracle[v];
}
```



Meltdown: Transiently encoding unauthorized memory

Unauthorized access → Transient out-of-order window → **Exception**

(discard architectural state)

Listing 1: x86 assembly.

```
meltdown:
  // %rdi: oracle
  // %rsi: secret_ptr

  movb (%rsi), %al
  shl $0xc, %rax
  movq (%rdi, %rax), %rdi
  retq
```

Listing 2: C code.

```
void meltdown(
        uint8_t *oracle,
        uint8_t *secret_ptr)
{
  uint8_t v = *secret_ptr;
  v = v * 0x1000;
  uint64_t o = oracle[v];
}
```

Meltdown: Transiently encoding unauthorized memory

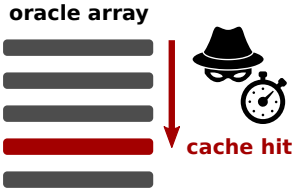
Unauthorized access → Transient out-of-order window → **Exception handler**

Listing 1: x86 assembly.

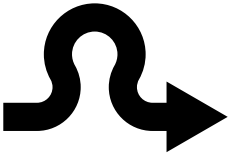
```
meltdown:
  // %rdi: oracle
  // %rsi: secret_ptr

  movb (%rsi), %al
  shl $0xc, %rax
  movq (%rdi, %rax), %rdi
  retq
```

Listing 2: C code.

```
void meltdown(
        uint8_t *oracle,
        uint8_t *secret_ptr)
{
  uint8_t v = *secret_ptr;
  v = v * 0x1000;
  uint64_t o = oracle[v];
}
```

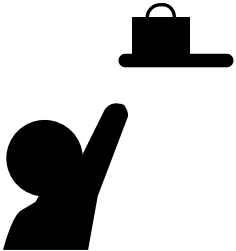


Mitigating Meltdown: Unmap kernel addresses from user space



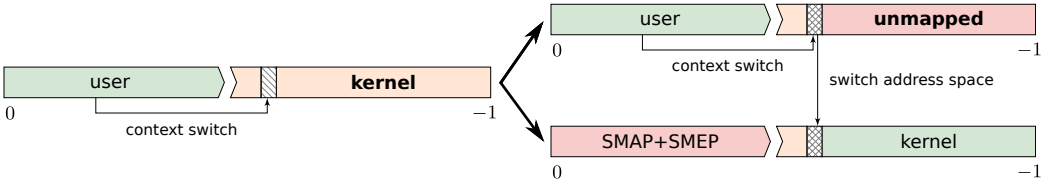
- OS software fix for faulty hardware (↔ future CPUs)

Mitigating Meltdown: Unmap kernel addresses from user space



- OS software fix for faulty hardware (↔ future CPUs)
- Unmap kernel from user *virtual address space*

→ Unauthorized physical addresses out-of-reach (˜cookie jar)



Gruss et al. "KASLR is dead: Long live KASLR", ESSoS 2017 [GLS+17]


inside™


inside™


inside™

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?~~

SPECTRE-LIKE FLAW UNDERMINES INTEL PROCESSORS' MOST SECURE ELEMENT

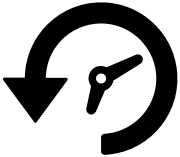
I'M SURE THIS WON'T BE THE LAST SUCH PROBLEM —

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

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

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

Building Foreshadow

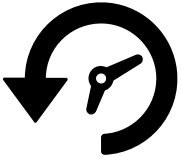


1. Cache secrets in L1

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

Foreshadow can read unmapped physical addresses from the cache (!)

Challenge: Reading unmapped secrets with Foreshadow

Untrusted world view

- Enclaved memory reads `0xFF`

Intra-enclave view

- Access enclaved + unprotected memory

Challenge: Reading unmapped secrets with Foreshadow

Untrusted world view

- Enclaved memory reads `0xFF`

Intra-enclave view

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

Challenge: Reading unmapped secrets with Foreshadow

Untrusted world view

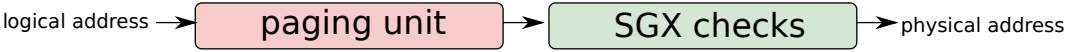
- Enclaved memory reads `0xFF`
- Meltdown "bounces back" (~ mirror)

Intra-enclave view

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

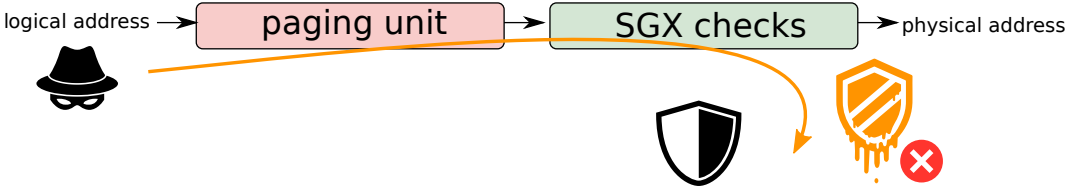
Building Foreshadow: Evade SGX abort page semantics

Note: SGX MMU sanitizes *untrusted* address translation



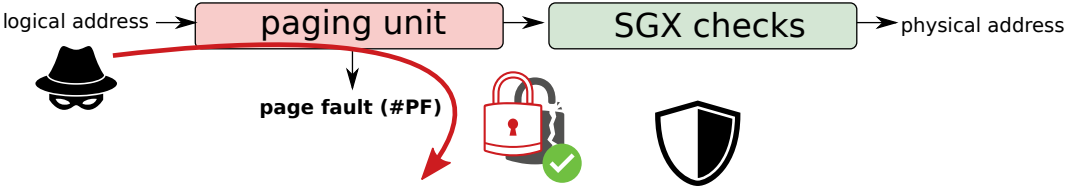
Building Foreshadow: Evade SGX abort page semantics

Meltdown: (Transient) accesses in non-enclave mode are dropped

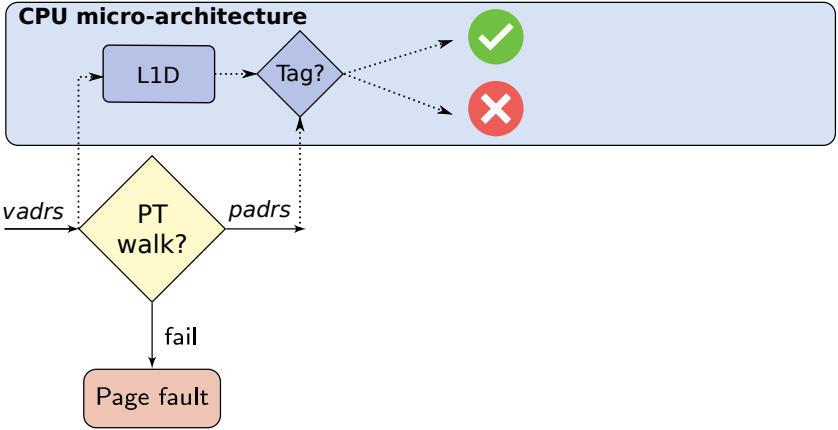


Building Foreshadow: Evade SGX abort page semantics

Foreshadow: Bypass abort page via *untrusted* page table

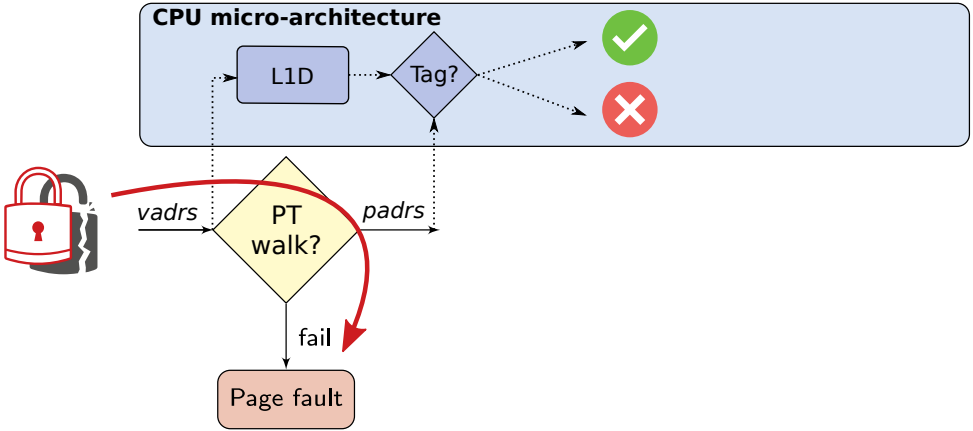


Foreshadow-NG: Breaking the virtual memory abstraction



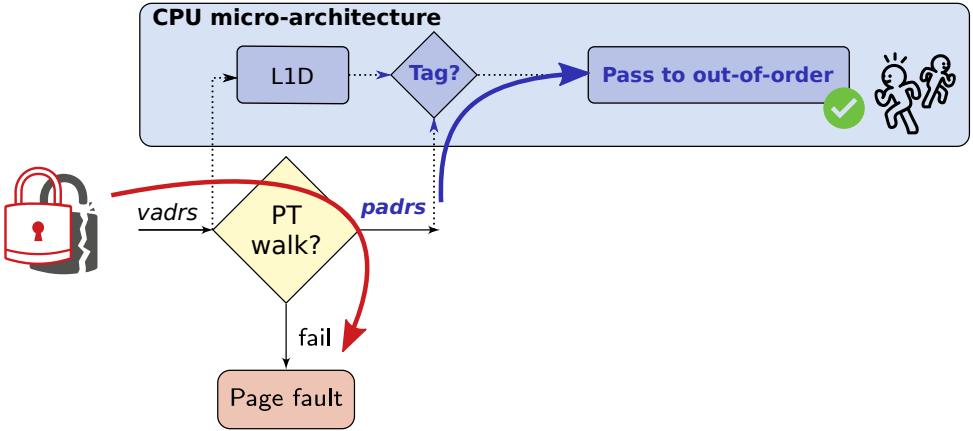
L1 cache design: Virtually-indexed, physically-tagged

Foreshadow-NG: Breaking the virtual memory abstraction



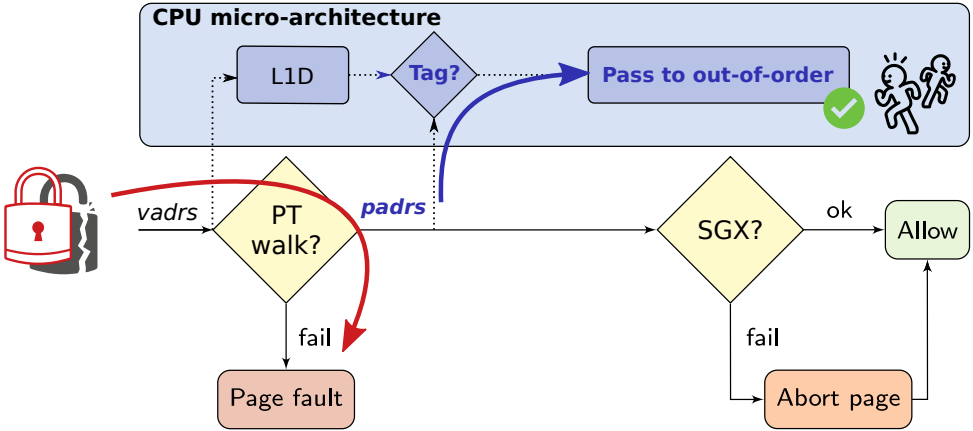
Page fault: Early-out address translation

Foreshadow-NG: Breaking the virtual memory abstraction



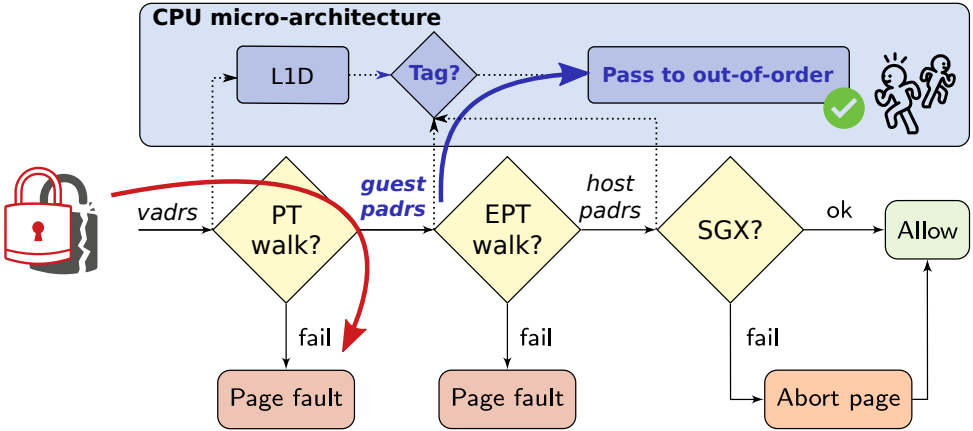
L1-Terminal Fault: match *unmapped physical address* (!)

Foreshadow-NG: Breaking the virtual memory abstraction



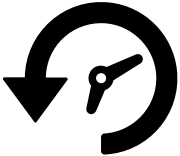
Foreshadow-SGX: bypass enclave isolation

Foreshadow-NG: Breaking the virtual memory abstraction



Foreshadow-VMM: bypass virtual machine isolation

Mitigating Foreshadow

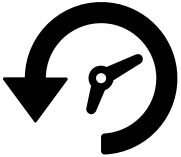


1. Cache secrets in L1

2. Unmap page table entry

3. Execute Meltdown

Mitigating Foreshadow



1. Cache secrets in L1

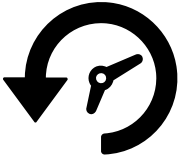
2. Unmap page table entry

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)

https://wiki.ubuntu.com/SecurityTeam/KnowledgeBase/L1TF

Mitigating Foreshadow

1. Cache secrets in L1

Intel microcode updates

2. Unmap page table entry

3. Execute Meltdown

$\Rightarrow$ **Flush L1** cache on enclave/VMM exit + **disable HyperThreading**

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

Mitigating Foreshadow/L1TF: Hardware-software cooperation

```
jo@gropius:~$ uname -svp
Linux #41~16.04.1-Ubuntu SMP Wed Oct 10 20:16:04 UTC 2018 x86_64

jo@gropius:~$ cat /proc/cpuinfo | grep "model name" -m1
model name      : Intel(R) Core(TM) i7 6500U CPU @ 2.50GHz

jo@gropius:~$ cat /proc/cpuinfo | egrep "meltdown|l1tf" -m1
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf

jo@gropius:~$ cat /sys/devices/system/cpu/vulnerabilities/meltdown | grep "Mitigation"
Mitigation: PTI

jo@gropius:~$ cat /sys/devices/system/cpu/vulnerabilities/l1tf | grep "Mitigation"
Mitigation: PTE Inversion; VMX: conditional cache flushes, SMT vulnerable

jo@gropius:~$
```

MELTDOWN

FORESHADOW

Some good news?

A lingering risk: Because Foreshadow, Spectre, and Meltdown are all hardware-based flaws, there's no guaranteed fix short of swapping out the chips. But security experts say the weaknesses are incredibly hard to exploit and that there's no evidence so far to suggest this year's chipocalypse has led to a hacking spree. Still, if your computer offers you an urgent software upgrade, be sure to take it immediately.

https://www.technologyreview.com/the-download/611879/intels-foreshadow-flaws-are-the-latest-sign-of-the-chipocalypse/

For the latest Intel security news, please visit security newsroom.

For all others, visit the Intel Security Center for the latest security information.

L1TF is a highly sophisticated attack method, and today, Intel is not aware of any reported real-world exploits.

https://www.intel.com/content/www/us/en/architecture-and-technology/l1tf.html

Some good news?

Azure confidential computing: Microsoft boosts security for cloud data

Microsoft is rolling out new secure enclave technology for protecting data in use.

By Liam Tung | October 26, 2017 -- 05:57 GMT (06:57 BST) | Topic: Cloud

https://www.zdnet.com/article/azure-confidential-computing-microsoft-boosts-security-for-cloud-data/

Some good news?

Microsoft Azure

$\longleftrightarrow$

Azure confidential computing: Microsoft boosts security for cloud data

Microsoft is rolling out new secure enclave technology for protecting data in use.

By Liam Tung | September 28, 2017 -- 05:57 GMT (06:57 BST) | Topic: Cloud

https://www.zdnet.com/article/azure-confidential-computing-microsoft-boosts-security-for-cloud-data/

Foreshadow fallout: Dismantling the SGX ecosystem

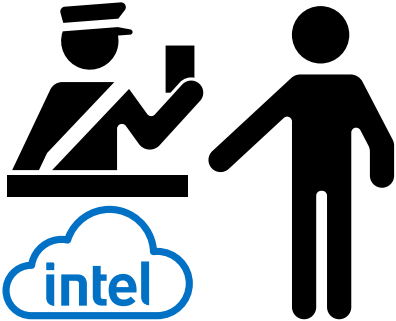
Remote attestation and secret provisioning

Challenge-response to prove **enclave identity**

Foreshadow fallout: Dismantling the SGX ecosystem

CPU-level key derivation

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



Foreshadow fallout: Dismantling the SGX ecosystem

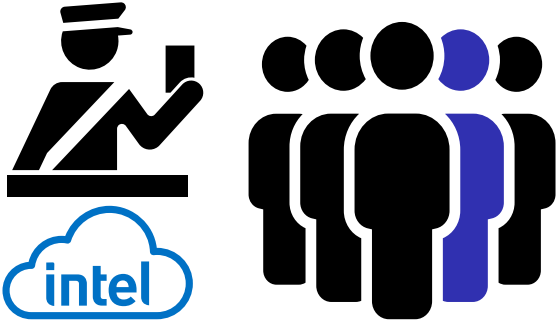
CPU-level key derivation

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

Foreshadow fallout: Dismantling the SGX ecosystem

Fully anonymous attestation

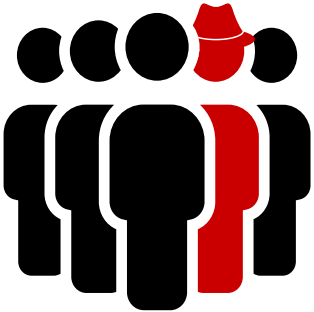
Intel Enhanced Privacy ID (EPID) **group signatures** ☺



Foreshadow fallout: Dismantling the SGX ecosystem

The dark side of anonymous attestation

Single **compromised EPID key** affects millions of devices . . . ☹



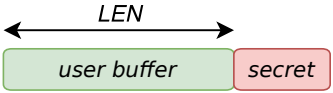
Foreshadow fallout: Dismantling the SGX ecosystem

EPID key extraction with Foreshadow

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


inside™
inside™
inside™

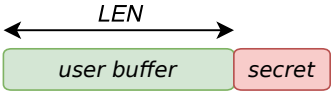
Spectre v1: Speculative buffer over-read




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

- Programmer *intention:* never access out-of-bounds memory

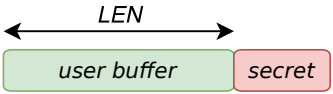
Spectre v1: Speculative buffer over-read




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

- Programmer *intention:* never access out-of-bounds memory
- Branch can be mistrained to speculatively (i.e., ahead of time) execute with $idx \geq LEN$ in the **transient world**

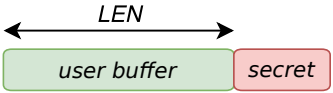
Spectre v1: Speculative buffer over-read




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

- Programmer *intention:* never access out-of-bounds memory
- Branch can be mistrained to speculatively (i.e., ahead of time) execute with $idx \geq LEN$ in the **transient world**
- Side-channels leak out-of-bounds secrets to the **real world**

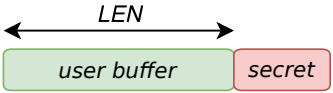
Mitigating Spectre v1: Inserting speculation barriers




```
if (idx < LEN)
{

  s = buffer[idx];
  t = lookup[s];
  ...
}
```

- Programmer *intention:* never access out-of-bounds memory

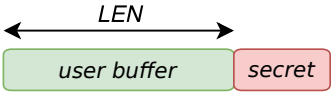
Mitigating Spectre v1: Inserting speculation barriers




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

- Programmer *intention:* never access out-of-bounds memory
- Insert **speculation barrier** to tell the CPU to halt the transient world until *idx* got evaluated ↔ performance ☹

Mitigating Spectre v1: Inserting speculation barriers




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

- Programmer *intention:* never access out-of-bounds memory
- Insert **speculation barrier** to tell the CPU to halt the transient world until *idx* got evaluated ↔ performance ☹
- Huge error-prone **manual effort,** no reliable automated compiler approaches yet...

index : kernel/git/torvalds/linux.git

Linux kernel source tree

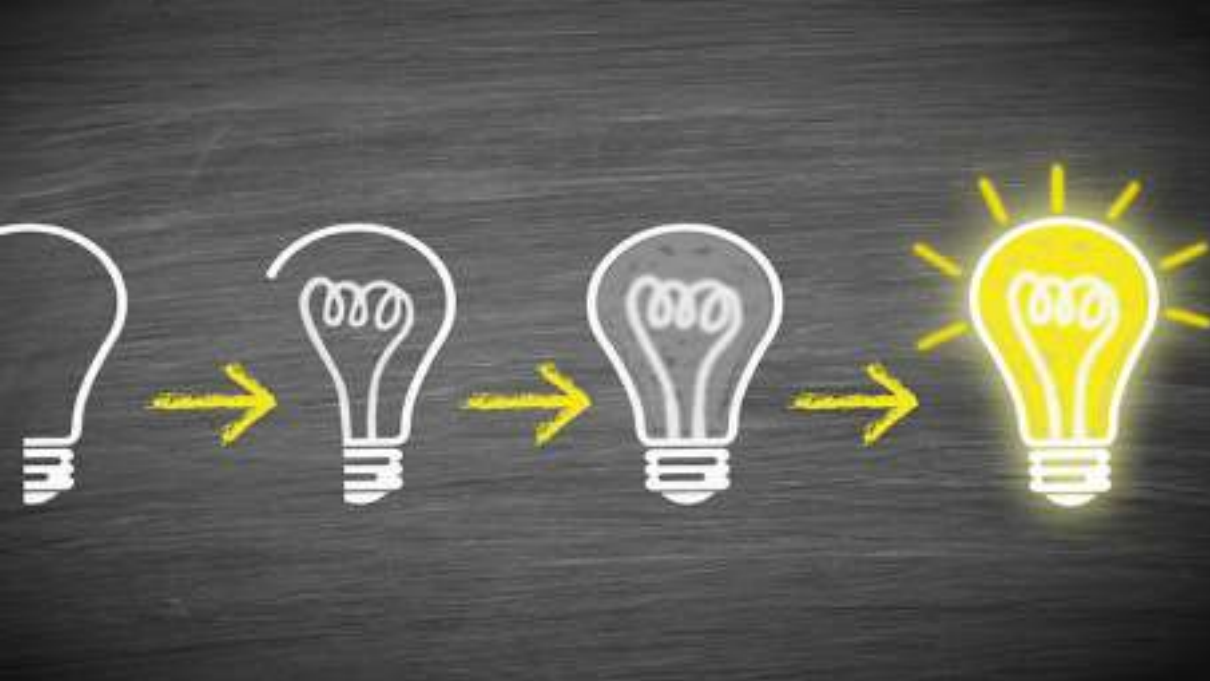
master switch

Linus Tor

about summary refs log tree commit diff stats

log msg Spectre v1 search

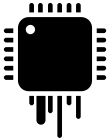
Age	Commit message (Expand)	Author	Files	Lines
3 days	Merge git://git.kernel.org/pub/scm/linux/kernel/git/davem/net	Linus Torvalds	56	-274/+795
4 days	vhost: Fix Spectre v1 vulnerability	Jason Wang	1	-0/+2
2018-10-19	Merge tag 'usb-4.19-final' of git://git.kernel.org/pub/scm/linux/kernel/git/g...	Greg Kroah-Hartman	7	-27/+65
2018-10-19	Merge git://git.kernel.org/pub/scm/linux/kernel/git/davem/net	Greg Kroah-Hartman	57	-187/+253
2018-10-19	Merge tag 'for-gkh' of git://git.kernel.org/pub/scm/linux/kernel/git/rdma/rdma	Greg Kroah-Hartman	2	-0/+6
2018-10-17	ptp: fix Spectre v1 vulnerability	Gustavo A. R. Silva	1	-0/+4
2018-10-17	usb: gadget: storage: Fix Spectre v1 vulnerability	Gustavo A. R. Silva	1	-0/+3
2018-10-16	RDMA/ucma: Fix Spectre v1 vulnerability	Gustavo A. R. Silva	1	-0/+3
2018-10-16	IB/ucm: Fix Spectre v1 vulnerability	Gustavo A. R. Silva	1	-0/+3
2018-09-25	Merge tag 'tty-4.19-rc6' of git://git.kernel.org/pub/scm/linux/kernel/git/gre...	Greg Kroah-Hartman	6	-7/+30
2018-09-18	tty: vt_ioctl: fix potential Spectre v1	Gustavo A. R. Silva	1	-0/+4
2018-09-14	Merge tag 'char-misc-4.19-rc4' of git://git.kernel.org/pub/scm/linux/kernel/g...	Linus Torvalds	10	-34/+73
2018-09-12	Merge tag 'pci-v4.19-fixes-1' of git://git.kernel.org/pub/scm/linux/kernel/gi...	Linus Torvalds	8	-25/+41
2018-09-12	misc: hmc6352: fix potential Spectre v1	Gustavo A. R. Silva	1	-0/+2
2018-09-11	switchtec: Fix Spectre v1 vulnerability	Gustavo A. R. Silva	1	-0/+4
2018-08-29	Merge tag 'hwmon-for-linus-v4.19-rc2' of git://git.kernel.org/pub/scm/linux/k...	Linus Torvalds	5	-12/+32
2018-08-26	hwmon: (nct6775) Fix potential Spectre v1	Gustavo A. R. Silva	1	-0/+2
2018-08-17	Merge tag 'drm-next-2018-08-17' of git://anongit.freedesktop.org/drm/drm	Linus Torvalds	44	-156/+346



Conclusions and take-away

https://foreshadowattack.eu/

- ⟹ New class of **transient execution** attacks
- ⟹ Importance of fundamental **side-channel research**
- ⟹ Security **cross-cuts** the system stack: hardware, hypervisor, kernel, compiler, application


References I

C. Canella, J. Van Bulck, M. Schwarz, M. Lipp, B. von Berg, P. Ortner, F. Piessens, D. Evtyushkin, and D. Gruss.
A systematic evaluation of transient execution attacks and defenses.
arXiv preprint arXiv:1811.05441, 2018.

D. Gruss, M. Lipp, M. Schwarz, R. Fellner, C. Maurice, and S. Mangard.
KASLR is dead: Long live KASLR.
In *International Symposium on Engineering Secure Software and Systems*, pp. 161–176. Springer, 2017.

P. Kocher, J. Horn, A. Fogh, , D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom.
Spectre attacks: Exploiting speculative execution.
In *Proceedings of the 40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019.

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 *Proceedings of the 27th USENIX Security Symposium (USENIX Security 18)*, 2018.

J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx.
Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution.
In *Proceedings of the 27th USENIX Security Symposium*. USENIX Association, August 2018.

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 `https://foreshadowattack.eu/`, 2018.