Transient Execution Attacks: Lessons from Spectre, Meltdown, and Foreshadow

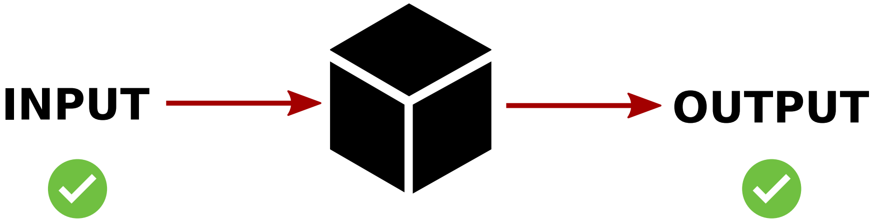
Jo Van Bulck

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

ISSE Brussels, November 6, 2018

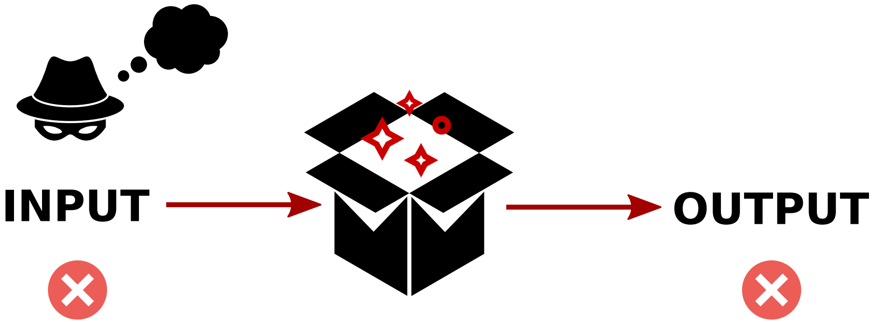
A primer on software security

Secure program: convert all input to *expected output*



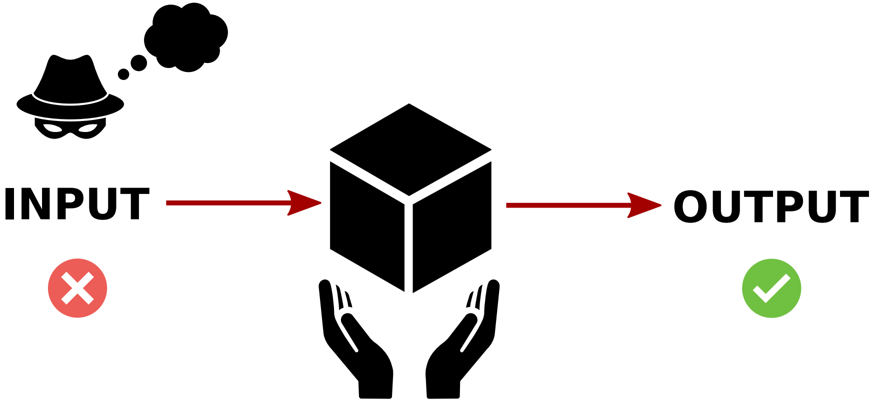
A primer on software security

Buffer overflow vulnerabilities: trigger *unexpected behavior*



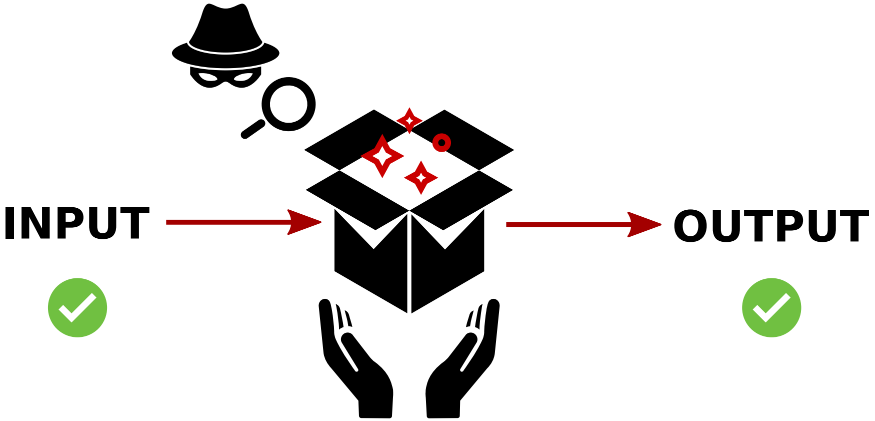
A primer on software security

Safe languages & formal verification: preserve *expected behavior*



A primer on software security

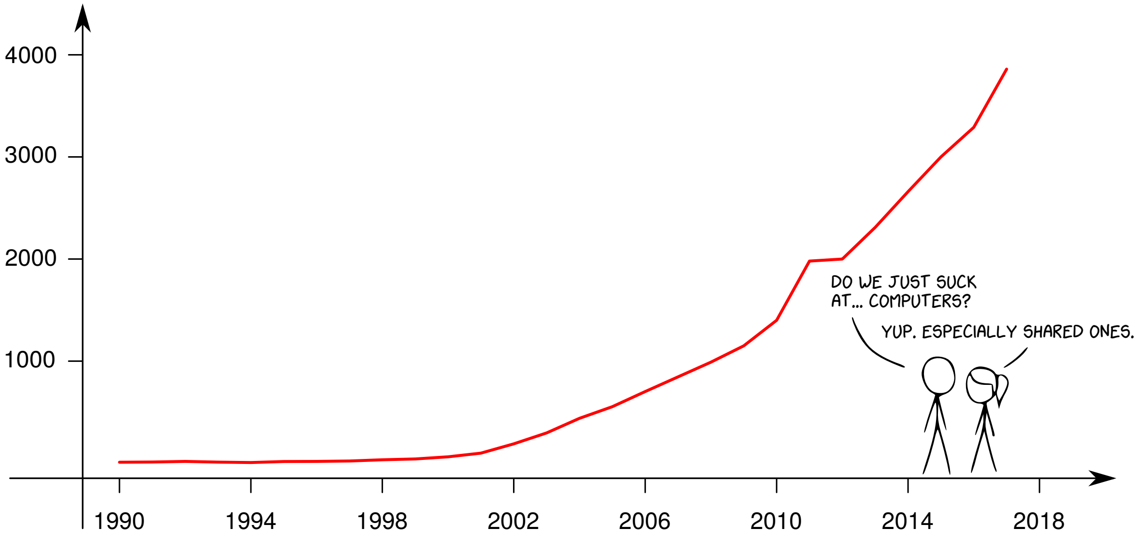
Side-channels: observe *side-effects* of the computation



VAULT DOOR
WEIGHT: 22 1/2 Tons
THICKNESS: 22 Inches
STEEL: 11 Layers of Special Cutting and Drill Resistant
LOCKS: 4 Hamilton Watch Movements for Time Locks

0
35
30

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

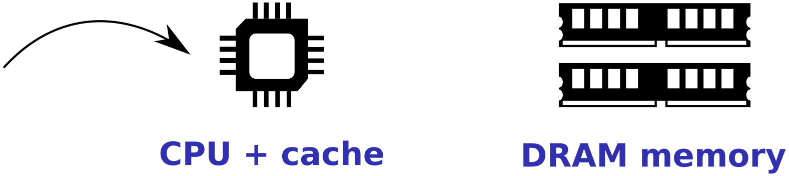


Based on `github.com/Pold87/academic-keyword-occurrence` and `xkcd.com/1938/`

CPU cache timing side-channel

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

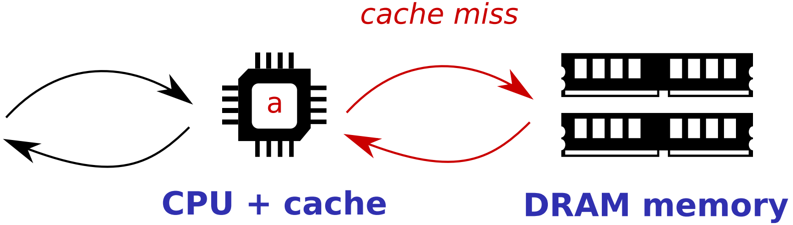
```
while true do
    maccess(&a);
endwh
```



CPU cache timing side-channel

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

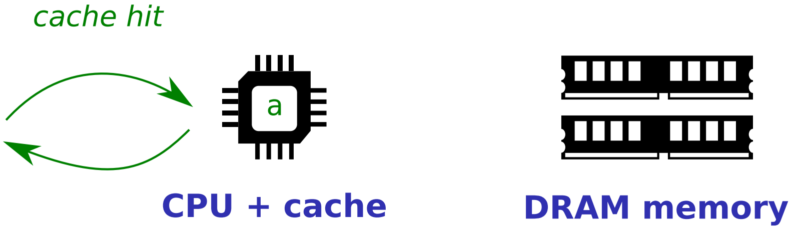
```
while true do
    maccess(&a);
endwh
```



CPU cache timing side-channel

Cache hit: No DRAM access required for subsequent uses

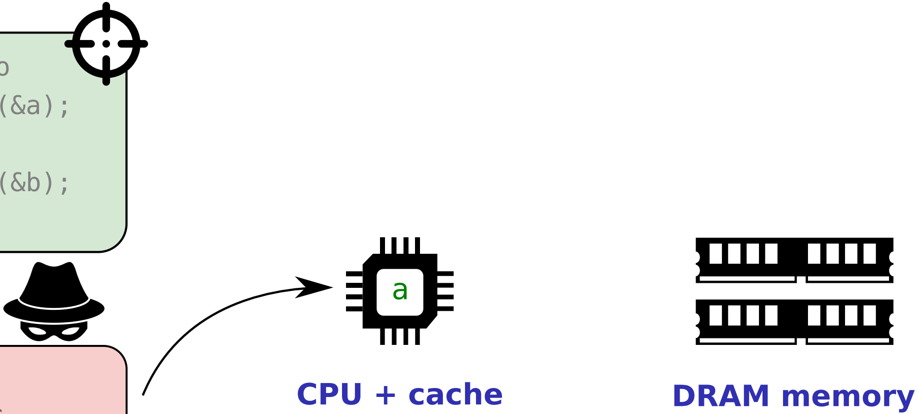
```
while true do
    maccess(&a);
endwh
```



Cache timing attacks in practice: Flush+Reload

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

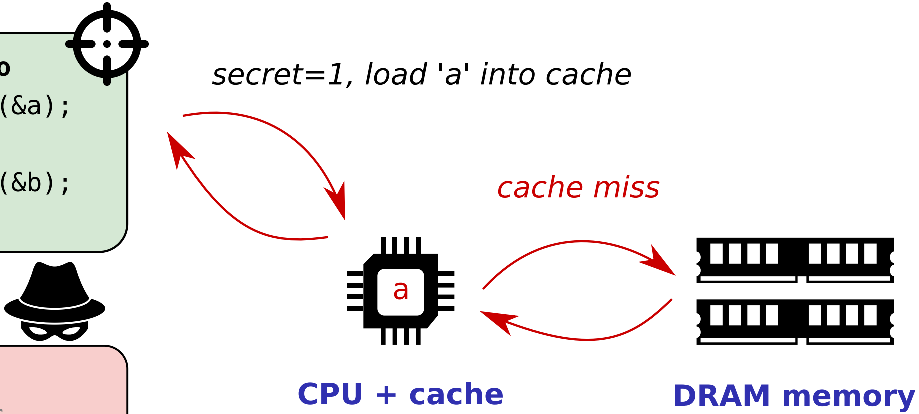
```
flush(&a);
start_timer
   maccess(&a);
end_timer
```



Cache timing attacks in practice: Flush+Reload

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

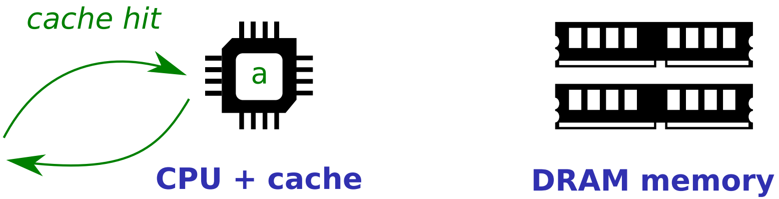
```
flush(&a);
start_timer
   maccess(&a);
end_timer
```



Cache timing attacks in practice: Flush+Reload

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

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

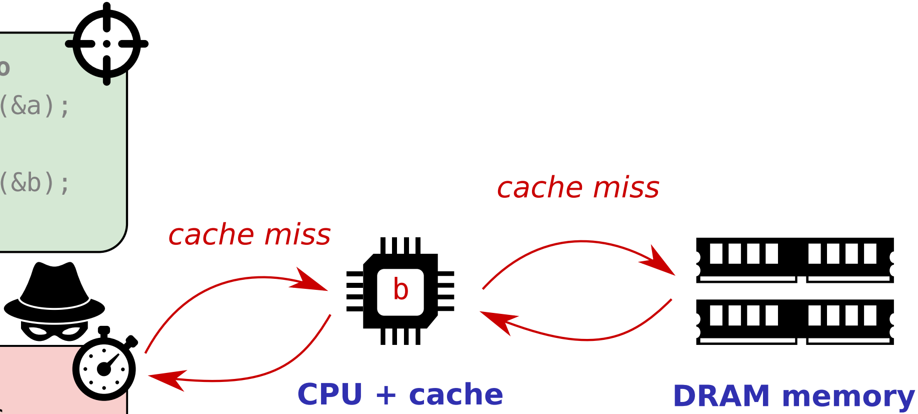


fast access(&a) → secret=1

Cache timing attacks in practice: Flush+Reload

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

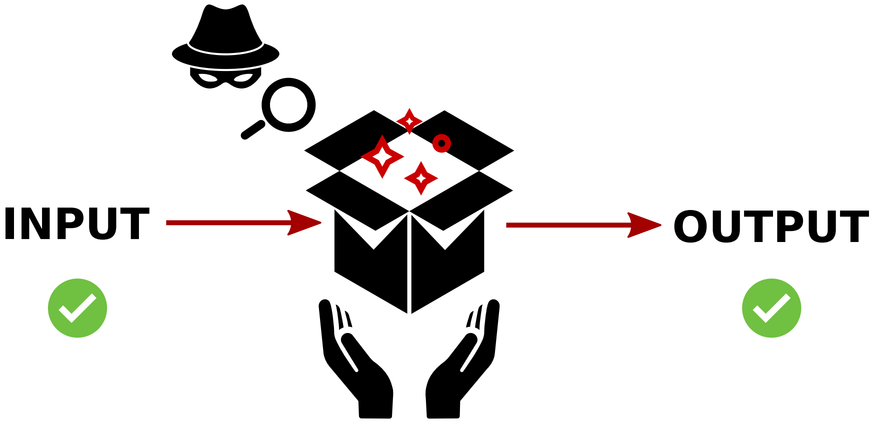
```
flush(&a);
start_timer
    maccess(&b);
end_timer
```



slow access(&b) → secret=1

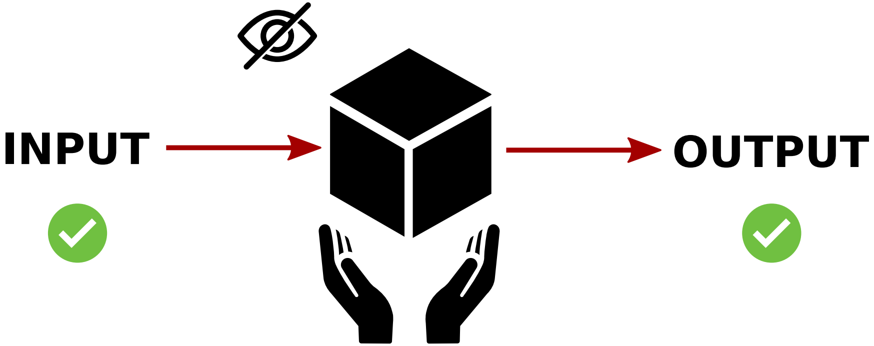
A primer on software security (revisited)

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



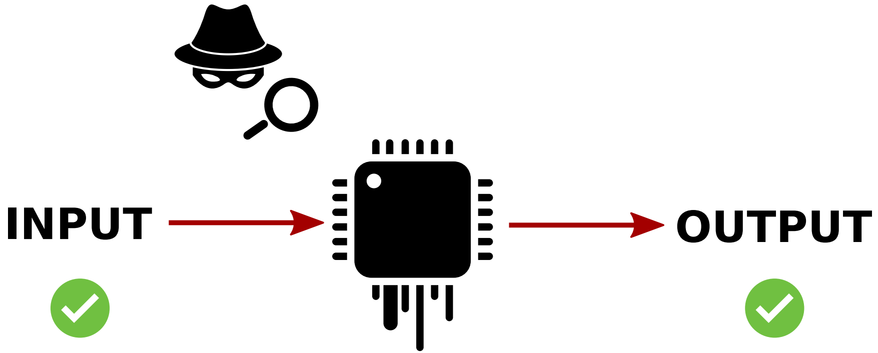
A primer on software security (revisited)

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



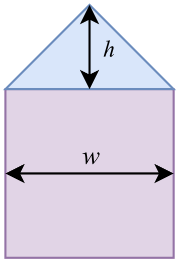
A primer on software security (revisited)

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

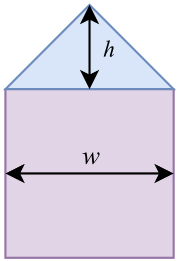


```
int area(int h, int w)
{
  int triangle = (w*h)/2;
  int square   = (w*w);
  return triangle + square;
}
```

Key **discrepancy:**

- Programmers write sequential instructions

Out-of-order and speculative execution



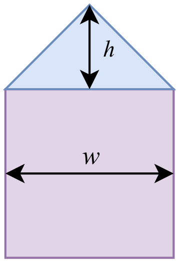
```
int area(int h, int w)
{
  int triangle = (w*h)/2;
  int square   = (w*w);
  return triangle + square;
}
```

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 (!)

Transient execution attacks: Welcome to the world of fun!

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

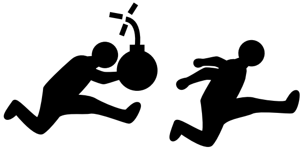
- Success → *commit* results to normal world ☺
- Fail → *discard* results, compute again in normal world ☹

Transient execution attacks: Welcome to the world of fun!

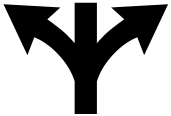
CPU executes ahead of time in **transient world**

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

Transient world (microarchitecture) may temp bypass architectural software intentions:



Delayed permission checks



Mispredict control flow

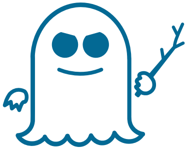
Transient execution attacks: Welcome to the world of fun!

Key finding of 2018

⇒ *transmit secrets from transient to normal world*

Transient world (microarchitecture) may temp bypass architectural software intentions:

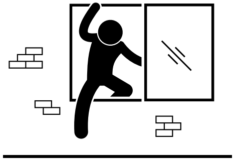
Delayed permission checks



Mispredict control flow

inside™
inside™
inside™

Meltdown: Transiently encoding unauthorized memory



Unauthorized access

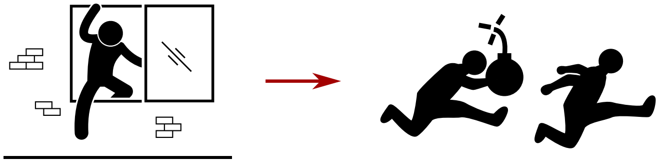
Listing 1: x86 assembly

```
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

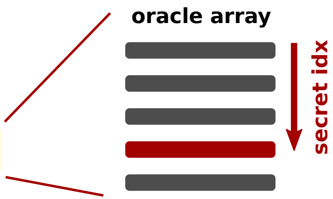
Transient out-of-order window

Listing 1: x86 assembly.

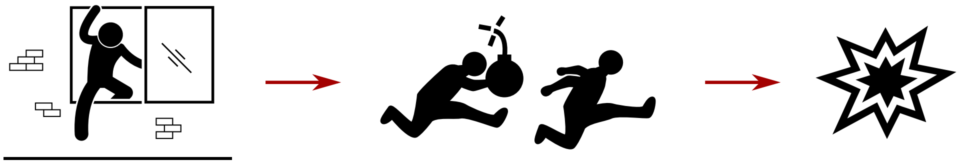
```
meltdown:
  // %rdi: oracle
  // %rsi: secret_ptr

  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)

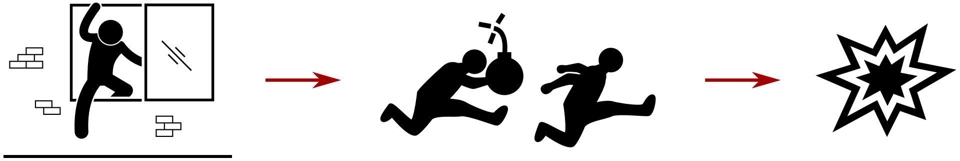
Listing 1: x86 assembly.

```
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

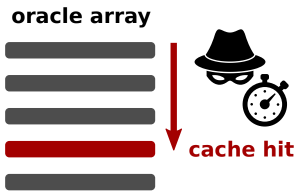
Exception handler

Listing 1: x86 assembly.

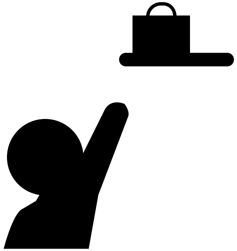
```
meltdown:
  // %rdi: oracle
  // %rsi: secret_ptr

  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

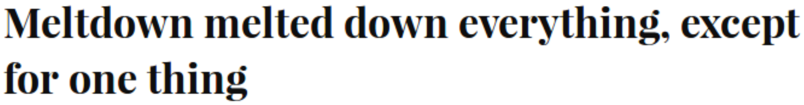


- OS software fix for faulty hardware ($\leftrightarrow$ future CPUs)
- Unmap kernel from user *virtual address space*

$\rightarrow$ Unauthorized physical addresses out-of-reach (˜cookie jar)

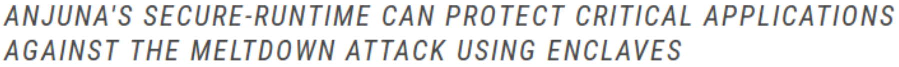
inside™
inside™
inside™

Rumors: Meltdown immunity for SGX enclaves?


Meltdown melted down everything, except for one thing

“[enclaves] remain protected and completely secure”

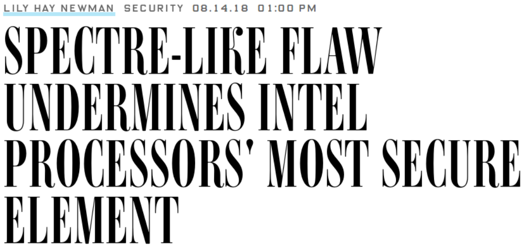
— *International Business Times, February 2018*


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

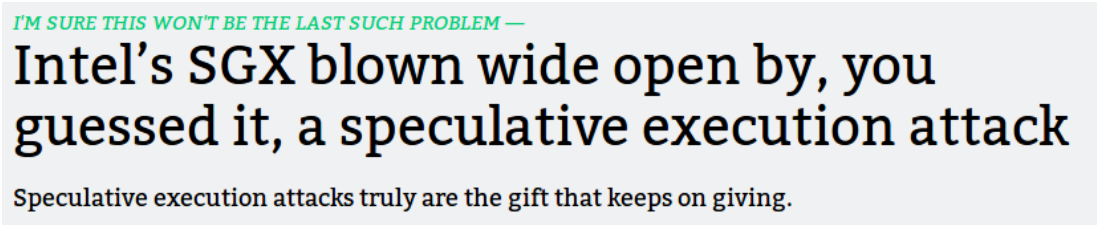
“[enclave memory accesses] redirected to an abort page, which has no value”

— *Anjuna Security, Inc., March 2018*

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


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

Building Foreshadow

1. Cache secrets in L1
2. Unmap page table entry
3. Execute Meltdown

Building Foreshadow

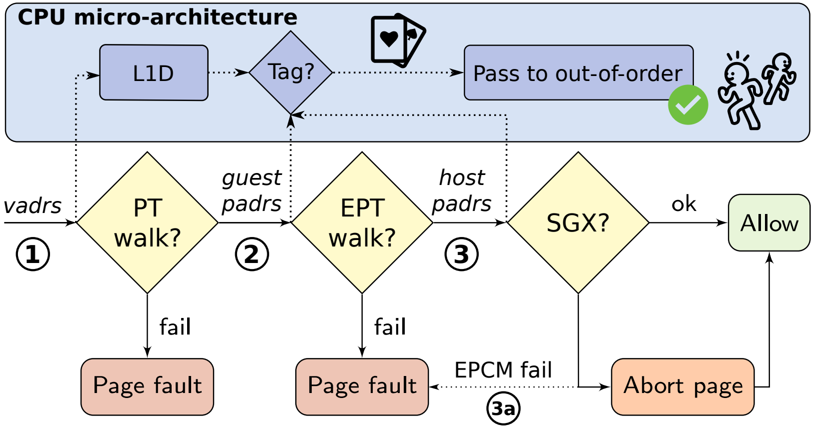
1. Cache secrets in L1

2. Unmap page table entry

3. Execute Meltdown

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

Foreshadow: Breaking the virtual memory abstraction



Arbitrary L1 cache read → bypass OS/hypervisor/enclave protection

Mitigating Foreshadow

1. Cache secrets in L1
2. Unmap page table entry
3. Execute Meltdown

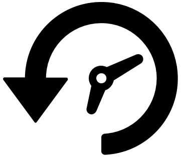
Mitigating Foreshadow

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
2. Unmap page table entry
3. Execute Meltdown

Intel microcode updates

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

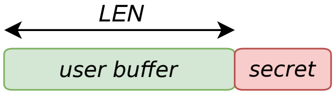
`https://software.intel.com/security-software-guidance/software-guidance/l1-terminal-fault`

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:~$ █
```

inside™
inside™
inside™

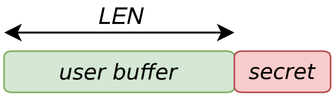
Spectre v1: Speculative buffer over-read



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

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

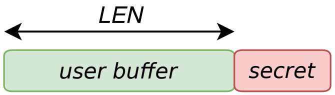
Spectre v1: Speculative buffer over-read



```
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**

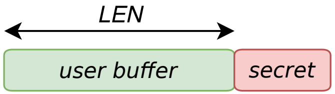
Spectre v1: Speculative buffer over-read



```
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**

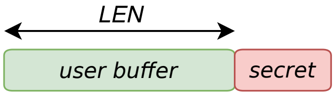
Mitigating Spectre v1: Inserting speculation barriers



```
if (idx < LEN)
{

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

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

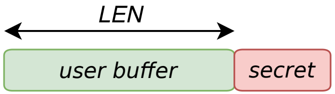
Mitigating Spectre v1: Inserting speculation barriers



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

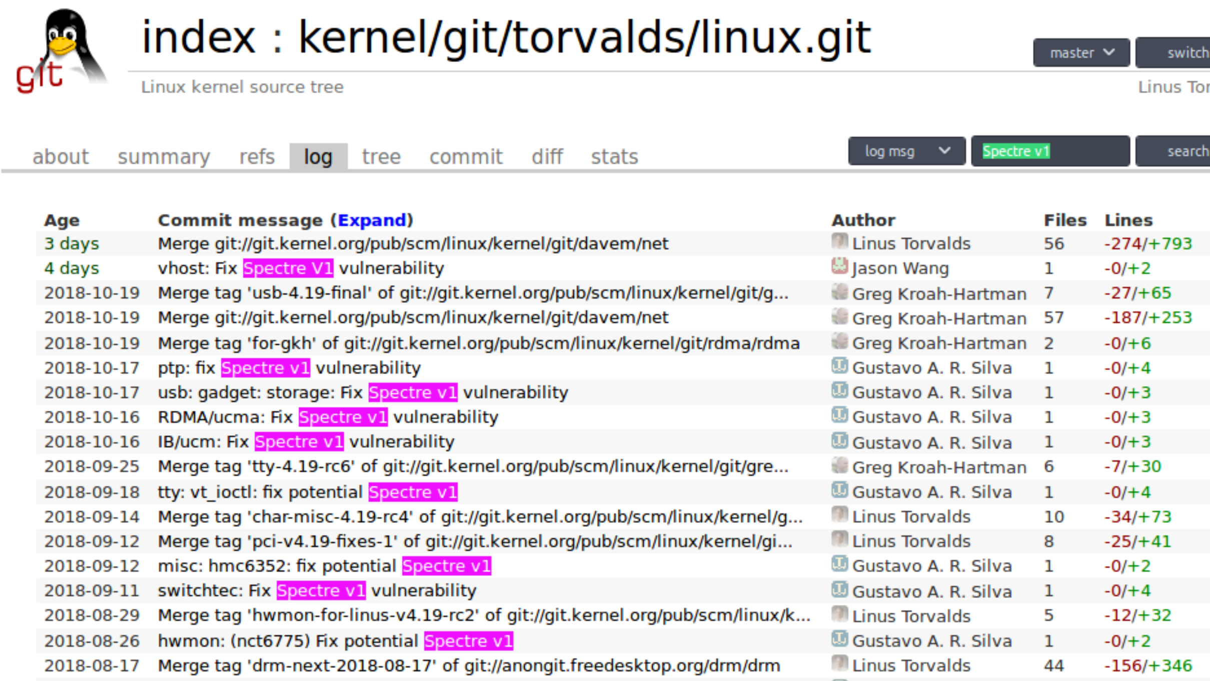
- Programmer *intention:* never access out-of-bounds memory
- Insert **speculation barrier** to tell the CPU to halt the transient world until *idx* got evaluated $\leftrightarrow$ performance ☹

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...



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/+793
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/

Hardware + software patches

Update your systems! (+ disable HyperThreading)

Conclusions and take-away

https://foreshadowattack.eu/

Hardware + software patches

Update your systems! (+ disable HyperThreading)

⇒ New class of **transient execution** attacks

⇒ Security **cross-cuts** the system stack: hardware, hypervisor, kernel, compiler, application

⇒ Importance of fundamental **side-channel research**

References I

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.

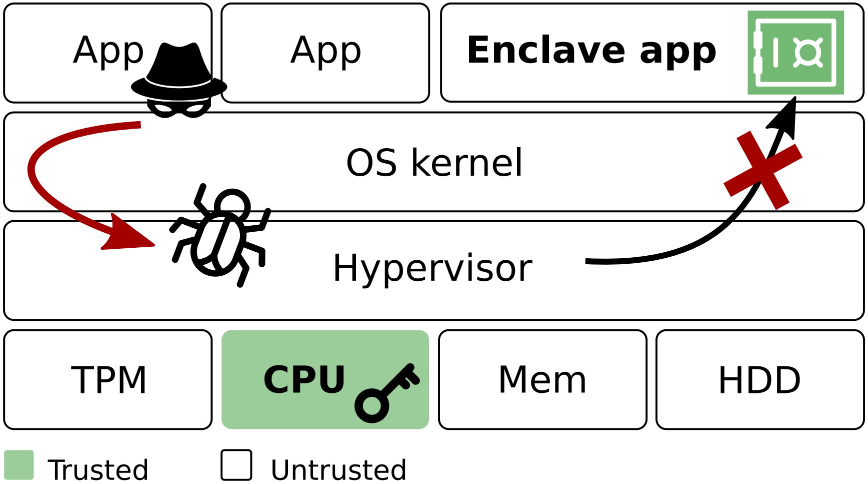
J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx.
Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution.
In *Proceedings of the 27th USENIX Security Symposium*. USENIX Association,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

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.

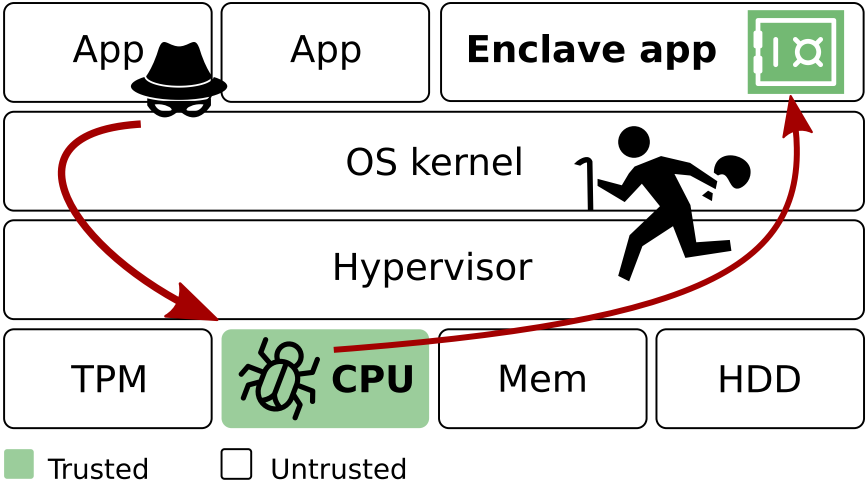
O. Weisse, J. Van Bulck, M. Minkin, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, R. Strackx, T. F. Wenisch, and Y. Yarom.
Foreshadow-NG: Breaking the virtual memory abstraction with transient out-of-order execution.
Technical Report https://foreshadowattack.eu/, 2018.

Y. Yarom and K. Falkner.
Flush+reload: A high resolution, low noise, L3 cache side-channel attack.
In *Proceedings of the 23rd USENIX Security Symposium*, pp. 719–732. USENIX Association, 2014.

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



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



Appendix: Challenge #1: Intel SGX abort page semantics

Untrusted world view

- Enclaved memory reads `0xFF`

Intra-enclave view

- Access enclaved + unprotected memory

Appendix: Challenge #1: Intel SGX abort page semantics

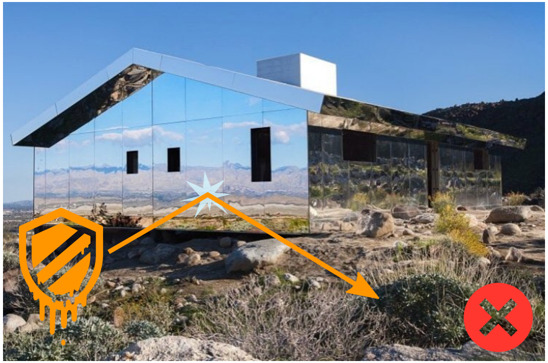
Untrusted world view

- Enclaved memory reads `0xFF`

Intra-enclave view

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

Appendix: 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