Leaky Processors: Lessons from Spectre, Meltdown, and Foreshadow

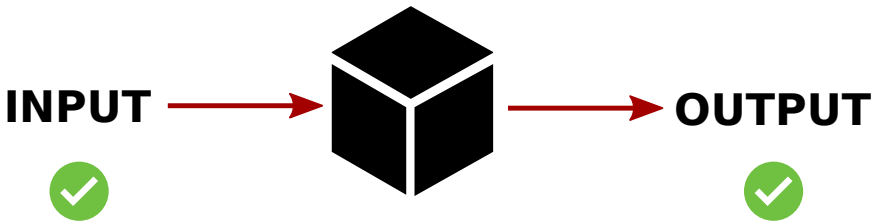
Jo Van Bulck

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

KU Leuven alumni forum, October 15, 2019

A primer on software security

Secure program: convert all input to *expected output*



Attacker-defender race

Security is an arms-race . . .

Attacker-defender race

Security is an arms-race . . . but also a positive story

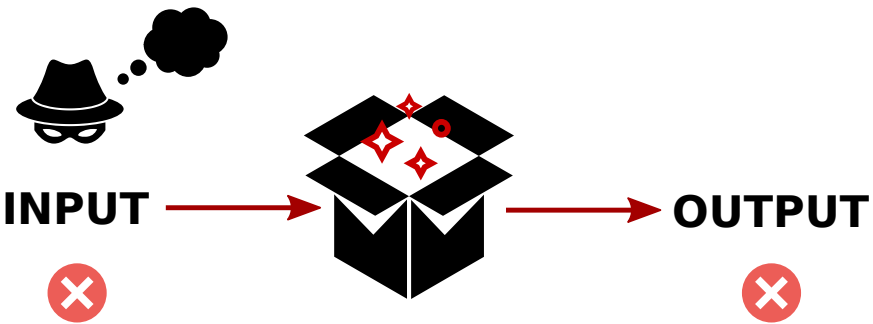
Attacker-defender race

Security is an arms-race . . . but also a positive story

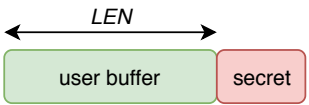
- Attacks are getting increasingly difficult; we are making progress!
- **Offensive research** keeps us on-par
  - . . . requires out-of-the-box thinking
  - . . . across the hardware-software boundary(!)

A primer on software security (revisited)

Buffer overflow vulnerabilities: trigger *unexpected behavior*



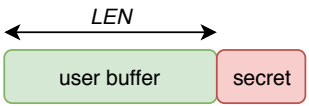
Buffer overflows: Triggering unexpected program behavior




```
void my_program (idx)
{
  s = buffer[idx];
  print(s);
  ...
}
```

- Program **intention:** never access out-of-bounds memory
- . . . but attacker may provide $idx \geq LEN$
- . . . and leak or modify secrets

Buffer overflows: Triggering unexpected program behavior



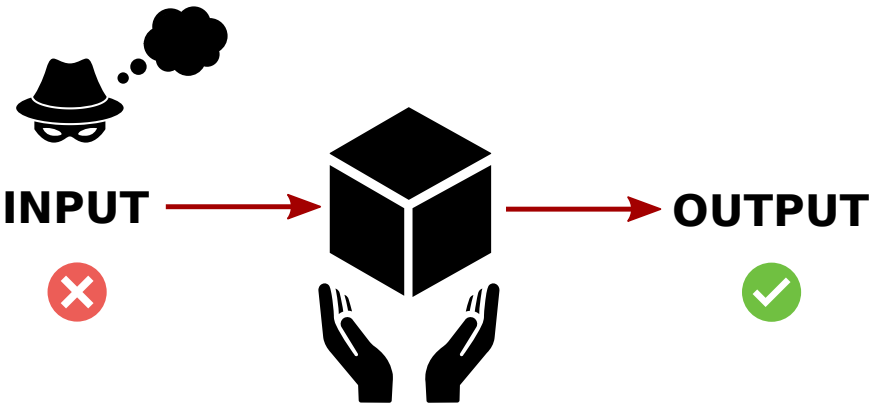

```
void my_program (idx)
{
  if (idx < LEN) {
    s = buffer[idx];
    print(s);
  }
  ...
}
```

- Program **intention:** never access out-of-bounds memory
- . . . but attacker may provide $idx \geq LEN$
- . . . and leak or modify secrets

$\Rightarrow$ include **defensive checks**

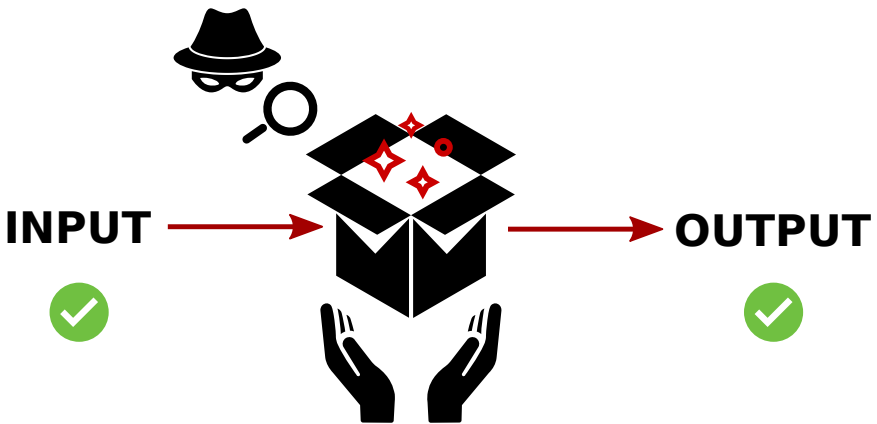
A primer on software security (revisited)

Safe languages & formal verification: preserve *expected behavior*



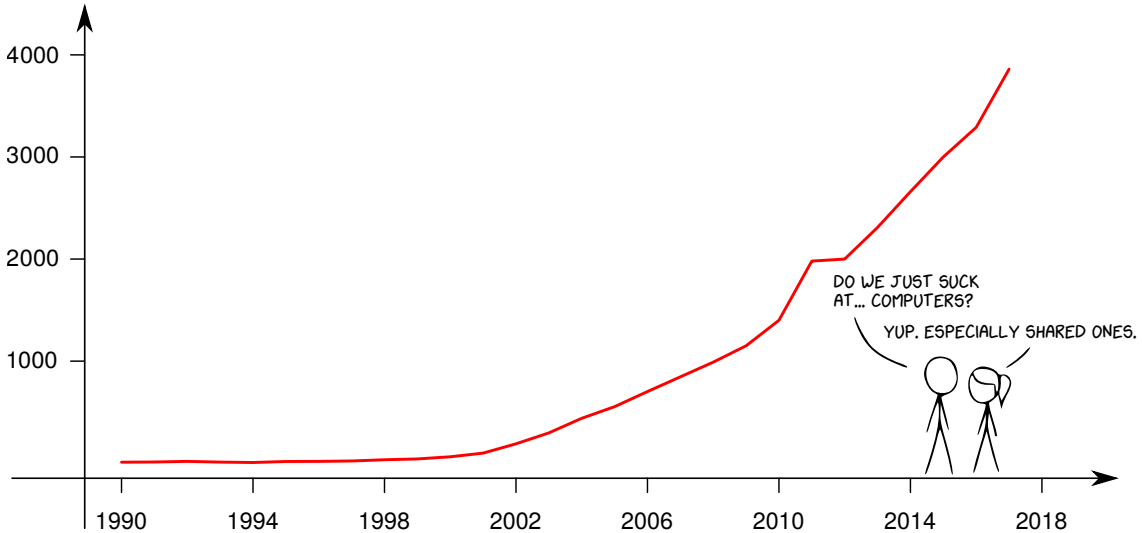
A primer on software security (revisited)

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



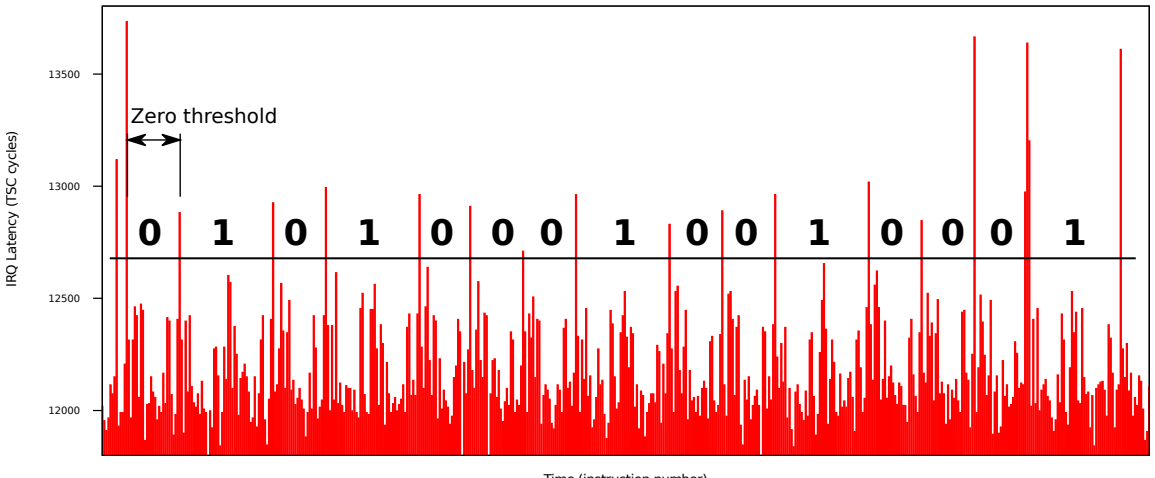

VAULT DOOR
WEIGHT: 22 1/2 Tons
THICKNESS: 22 Inches
STEEL: 11 Layers of Special Cutting and Drill Resistant
LOCKS: 4 Hamilton Watch Movements for Time Locks

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



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

Microarchitectural timing leaks in practice

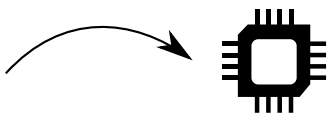


Van Bulck et al. "Nemesis: Studying Microarchitectural Timing Leaks in Rudimentary CPU Interrupt Logic", CCS 2018

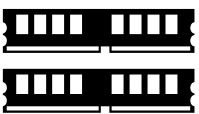
CPU cache timing side-channel

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

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



CPU + cache

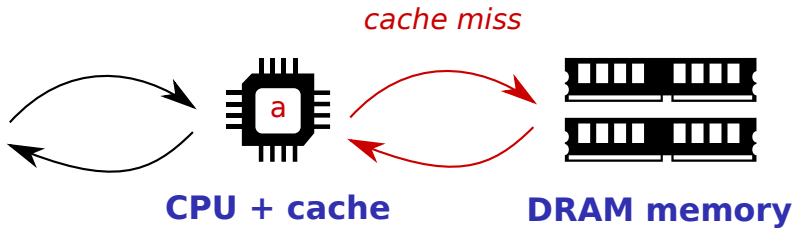


DRAM memory

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
```

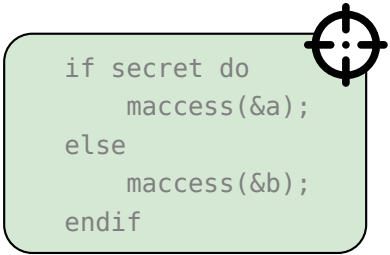
cache hit

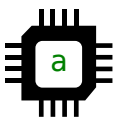
CPU + cache

DRAM memory

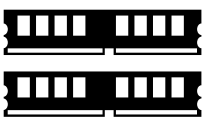

WHAT COULD POSSIBLY
38
GO WRONG?
imgflip.com

Cache timing attacks in practice: Flush+Reload



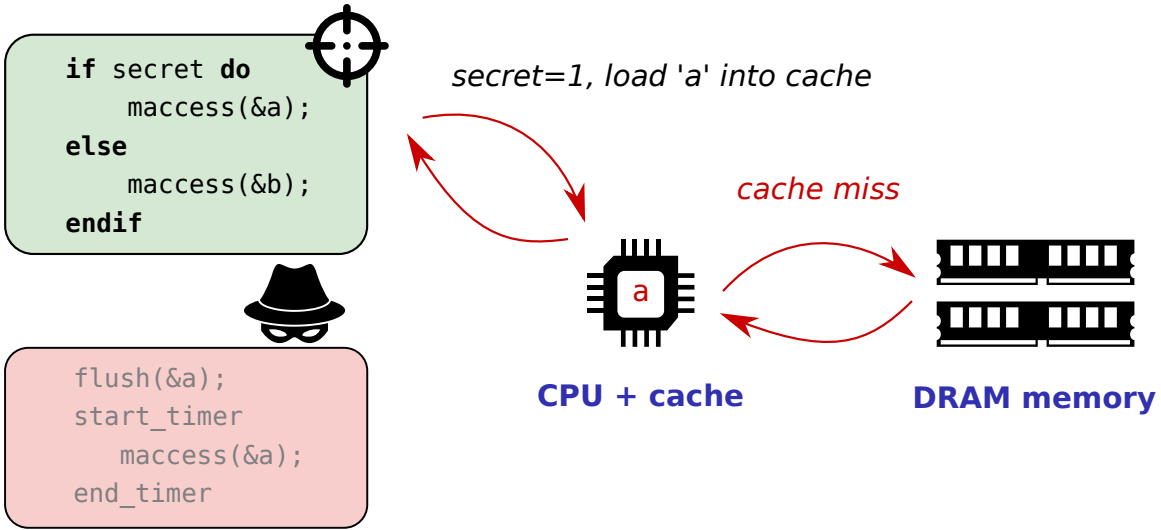


CPU + cache

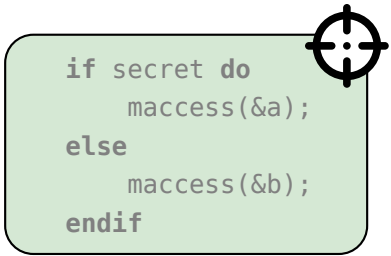


DRAM memory

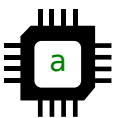
Cache timing attacks in practice: Flush+Reload



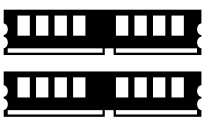
Cache timing attacks in practice: Flush+Reload



cache hit



CPU + cache



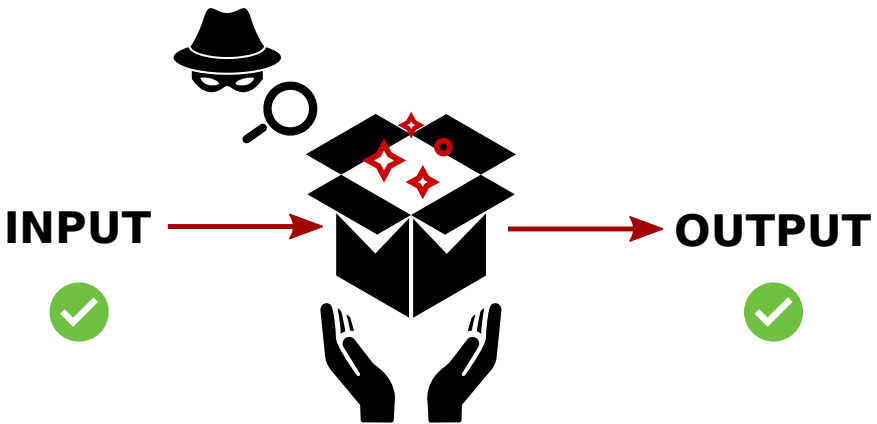
DRAM memory

fast access(&a) → secret=1


SHARING IS NOT CARING
SHARING IS LOSING YOUR STUFF TO OTHERS
imgflip.com

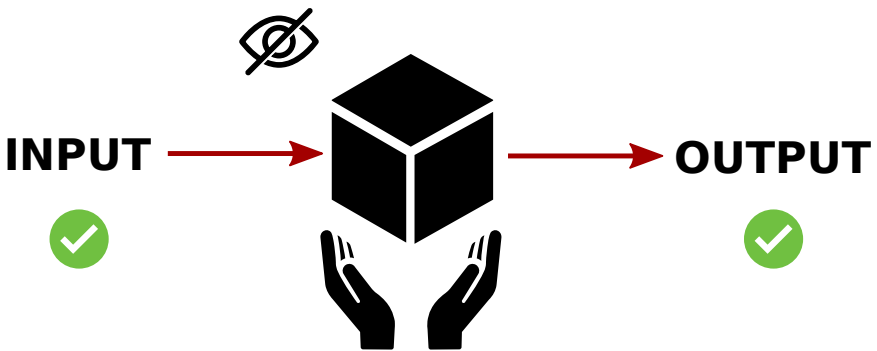
A primer on software security (revisited)

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



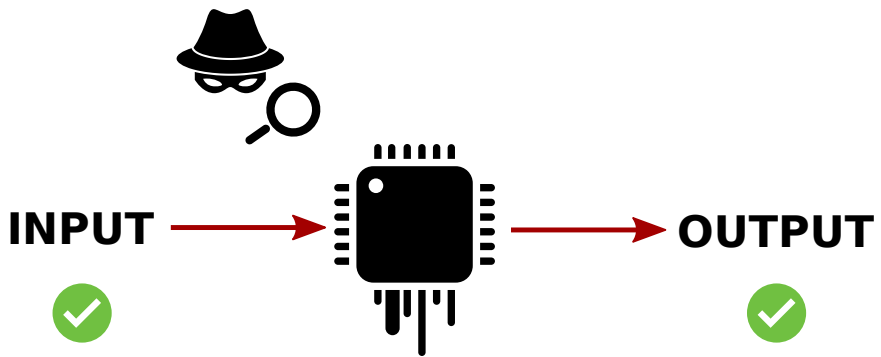
A primer on software security (revisited)

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



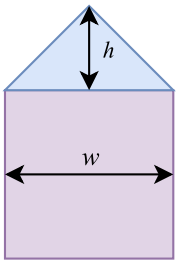
A primer on software security (revisited)

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

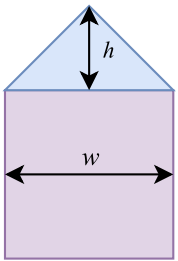



```
int area(int h, int w)
{
  int triangle = (w*h)/2;
  int square   = (w*w);
  return triangle + square;
}
```

Key **discrepancy:**

- Programmers write sequential instructions

Out-of-order and speculative execution



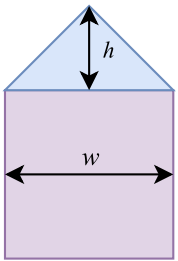

```
int area(int h, int w)
{
    int triangle = (w*h)/2;
    int square   = (w*w);
    return triangle + square;
}
```

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

$\Rightarrow$ *Speculatively execute instructions ahead of time*

Best-effort: What if triangle fails?

$\rightarrow$ Commit in-order, roll-back square

$\ldots$ 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 <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*

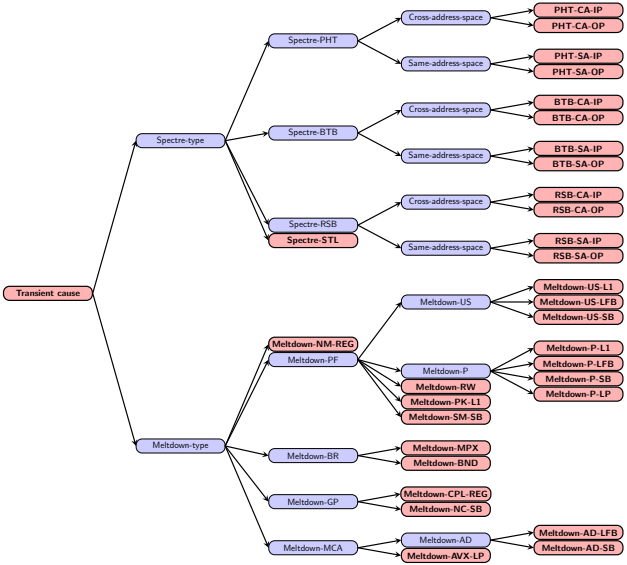
Transient world (microarchitecture) may temp bypass architectural software intentions:

CPU access control bypass

Speculative buffer overflow/ROP

The transient-execution zoo

https://transient.fail



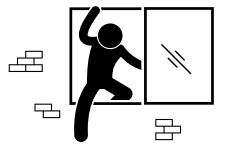
Canella et al. “A systematic evaluation of transient execution attacks and defenses”, USENIX Security 2019


inside™


inside™


inside™

Meltdown: Transiently encoding unauthorized memory



Unauthorized access

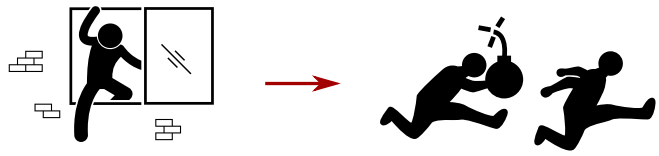
Listing 1: x86 assembly

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

Listing 2: C code.

```
1  void meltdown(
2         uint8_t *oracle,
3         uint8_t *secret_ptr)
4  {
5    uint8_t v = *secret_ptr;
6    v = v * 0x1000;
7    uint64_t o = oracle[v];
8  }
```

Meltdown: Transiently encoding unauthorized memory



Unauthorized access

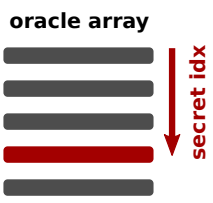
Transient out-of-order window

Listing 1: x86 assembly.

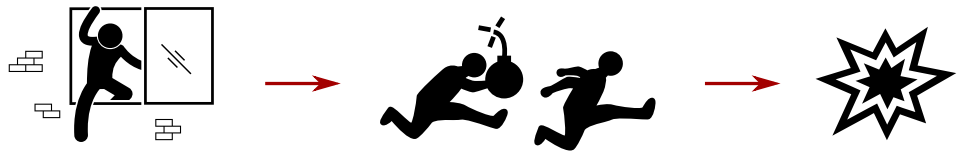
```
1 meltdown:
2     // %rdi: oracle
3     // %rsi: secret_ptr
4
5     movb (%rsi), %al
6     shl $0xc, %rax
7     movq (%rdi, %rax), %rdi
8     retq
```

Listing 2: C code.

```
1 void meltdown(
2         uint8_t *oracle,
3         uint8_t *secret_ptr)
4 {
5     uint8_t v = *secret_ptr;
6     v = v * 0x1000;
7     uint64_t o = oracle[v];
8 }
```



Meltdown: Transiently encoding unauthorized memory



Unauthorized access

Transient out-of-order window

Exception
(discard architectural state)

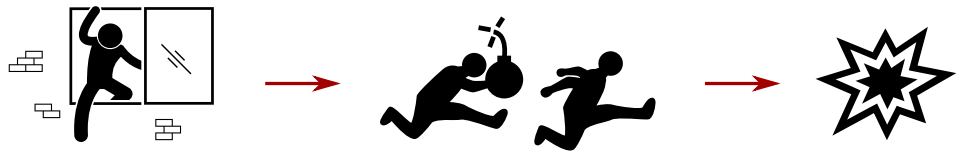
Listing 1: x86 assembly.

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

Listing 2: C code.

```
1 void meltdown(
2         uint8_t *oracle,
3         uint8_t *secret_ptr)
4 {
5     uint8_t v = *secret_ptr;
6     v = v * 0x1000;
7     uint64_t o = oracle[v];
8 }
```

Meltdown: Transiently encoding unauthorized memory



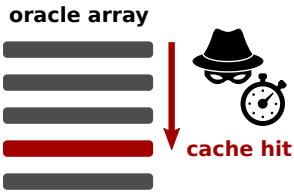
Unauthorized access | Transient out-of-order window | **Exception handler**

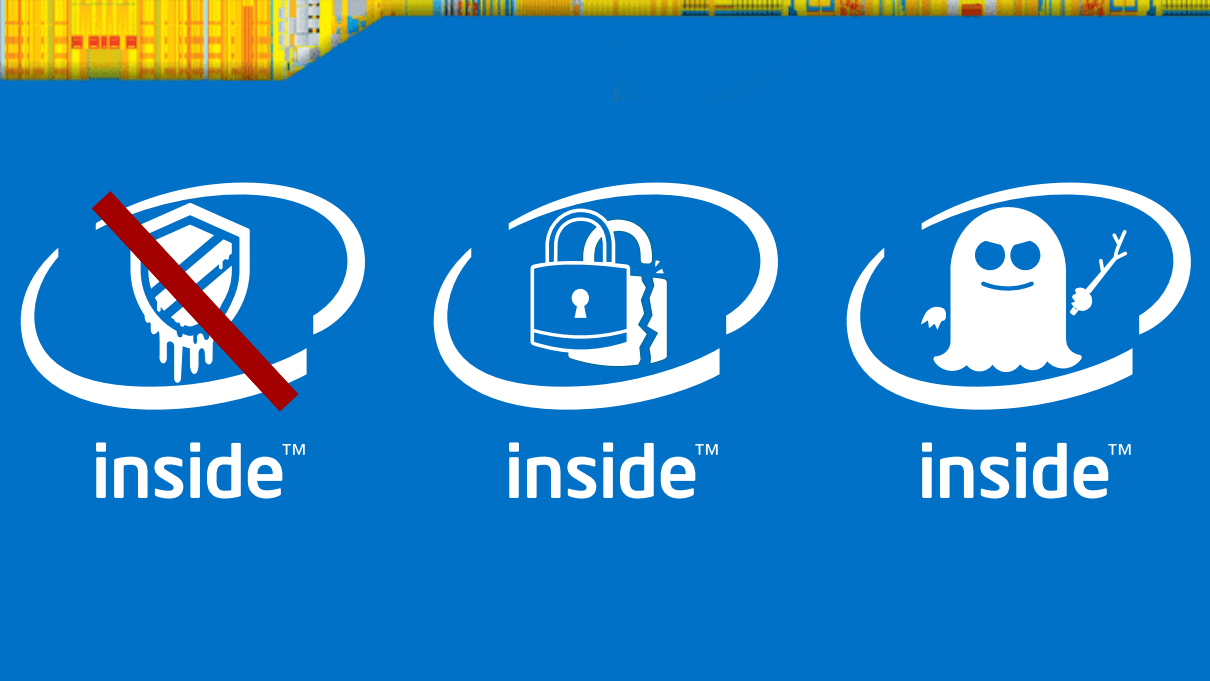
Listing 1: x86 assembly.

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

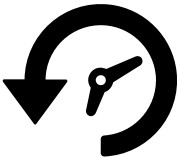
Listing 2: C code.

```
1 void meltdown(
2         uint8_t *oracle,
3         uint8_t *secret_ptr)
4 {
5     uint8_t v = *secret_ptr;
6     v = v * 0x1000;
7     uint64_t o = oracle[v];
8 }
```




inside™
inside™
inside™

Building Foreshadow

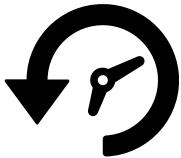


1. Cache secrets in L1

2. Unmap page table entry

3. Execute Meltdown

Building Foreshadow



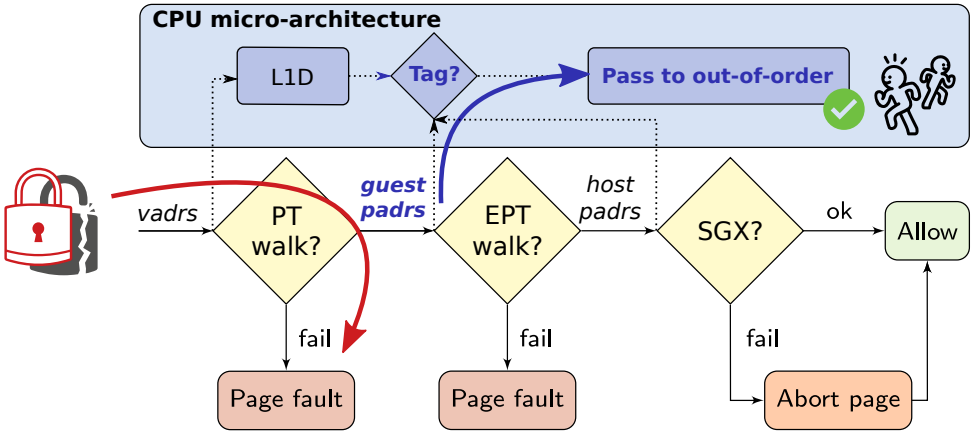
1. Cache secrets in L1

2. Unmap page table entry

3. Execute Meltdown

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

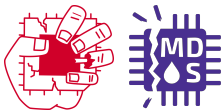
Foreshadow: Breaking the virtual memory abstraction



L1-Terminal Fault: bypass virtual machine and SGX isolation(!)

Mitigating Meltdown/Foreshadow: Hardware-software cooperation

```
jo@sgx-dsn:~$ uname -svp
Linux #74~16.04.1-Ubuntu SMP Wed Sep 18 09:51:44 UTC 2019 x86_64
jo@sgx-dsn:~$ cat /proc/cpuinfo | grep "model name" -m1
model name      : Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz
jo@sgx-dsn:~$ cat /proc/cpuinfo | grep "bugs" -m1
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds swapgs
jo@sgx-dsn:~$ ls /sys/devices/system/cpu/vulnerabilities/
l1tf  mds  meltdown  spec_store_bypass  spectre_v1  spectre_v2
jo@sgx-dsn:~$ cat /sys/devices/system/cpu/vulnerabilities/* | grep "Mitigation"
Mitigation: PTE Inversion; VMX: conditional cache flushes, SMT disabled
Mitigation: Clear CPU buffers; SMT disabled
Mitigation: PTI
Mitigation: Speculative Store Bypass disabled via prctl and seccomp
Mitigation: usercopy/swapgs barriers and __user pointer sanitization
Mitigation: Full generic retpoline, IBPB: conditional, IBRS_FW, RSB filling
jo@sgx-dsn:~$
```

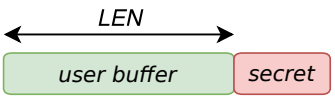



inside™


inside™


inside™

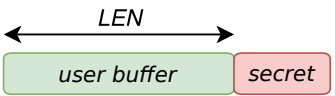
Spectre v1: Speculative buffer over-read




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

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

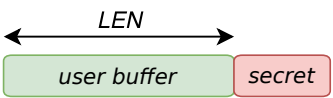
Spectre v1: Speculative buffer over-read




```
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)
{
  asm("lfence\n\t");
  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**
- Insert explicit **speculation barriers** to tell the CPU to halt the transient world...

Age	Commit message (Expand)	Author	Files	Lines
13 days	arm64: errata: Update stale comment	Thierry Reding	1	-2/+2
2019-09-18	HID: core: fix dmesg flooding if report field larger than 32bit	Joshua Clayton	1	-2/+2
2019-09-11	Merge tag 'mac80211-next-for-davem-2019-09-11' of git://git.kernel.org/pub/sc...	David S. Miller	19	-161/+274
2019-09-11	nl80211: Fix possible Spectre-v1 for CQM RSSI thresholds	Masashi Honma	1	-1/+3
2019-09-02	ALSA: hda/realtek - Add quirk for HP Pavilion 15	Sam Bazley	1	-0/+1
2019-08-28	Documentation/process: Embargoed hardware security issues	Thomas Gleixner	2	-0/+280
2019-08-27	powerpc/64s: support nospectre_v2 cmdline option	Christopher M. Riedl	1	-3/+16
2019-08-20	tools arch x86: Sync asm/cpufeatures.h with the with the kernel	Arnaldo Carvalho de Melo	1	-0/+3
2019-08-13	ALSA: hda/realtek - Add quirk for HP Envy x360	Takashi Iwai	1	-0/+1
2019-08-09	Merge branch 'for-linus' of git://git.kernel.org/pub/scm/linux/kernel/git/dto...	Linus Torvalds	7	-44/+56
2019-08-06	Merge branch 'x86/grand-schemozzle' of git://git.kernel.org/pub/scm/linux/ker...	Linus Torvalds	7	-40/+246
2019-08-03	Documentation: Add swapgs description to the Spectre v1 documentation	Josh Poimboeuf	1	-8/+80
2019-08-02	atm: iphase: Fix Spectre v1 vulnerability	Gustavo A. R. Silva	1	-2/+6
2019-08-02	Merge tag 'for-linus' of git://git.kernel.org/pub/scm/linux/kernel/git/rdma/rdma	Linus Torvalds	13	-83/+124
2019-08-01	IB/hfi1: Fix Spectre v1 vulnerability	Gustavo A. R. Silva	1	-0/+2
2019-08-01	IB/core: Add mitigation for Spectre V1	Luck, Tony	1	-1/+5
2019-07-28	Merge branch master from git://git.kernel.org/pub/scm/linux/kernel/git/torval...	Thomas Gleixner	12696	-527483/+1099601
2019-07-25	Input: synaptics - enable RMI mode for HP Spectre X360	Dmitry Torokhov	1	-0/+1
2019-07-22	x86: Remove X86_FEATURE_MFENCE_RDTSC	Josh Poimboeuf	6	-42/+8
2019-07-17	x86/entry/64: Use JMP instead of JMPQ	Josh Poimboeuf	1	-1/+1
2019-07-12	Merge tag 'for-linus' of git://git.kernel.org/pub/scm/virt/kvm/kvm	Linus Torvalds	92	-1432/+2693
2019-07-09	Merge tag 'docs-5.3' of git://git.lwn.net/linux	Linus Torvalds	416	-8581/+12159
2019-07-09	x86/speculation: Enable Spectre v1 swapgs mitigations	Josh Poimboeuf	2	-13/+110
2019-07-09	x86/speculation: Prepare entry code for Spectre v1 swapgs mitigations	Josh Poimboeuf	3	-3/+37
2019-07-08	Merge branch 'x86-pti-for-linus' of git://git.kernel.org/pub/scm/linux/kernel...	Linus Torvalds	3	-5/+12
2019-07-05	arm64: KVM: Propagate full Spectre v2 workaround state to KVM guests	Andre Przywara	5	-11/+56
2019-06-27	x86/tls: Fix possible spectre-v1 in do_get_thread_area()	Dianzhang Chen	1	-2/+7
2019-06-27	x86/ptrace: Fix possible spectre-v1 in ptrace_get_debugreg()	Dianzhang Chen	1	-1/+4
2019-06-26	Documentation: Add section about CPU vulnerabilities for Spectre	Tim Chen	3	-0/+700

Conclusions and take-away

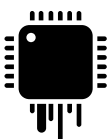
https://transient.fail/

⇒ Security is an **arms-race**, importance of fundamental offensive research

⇒ New class of **side-channel** and **transient-execution** attacks

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



References I

C. Canella, D. Genkin, L. Giner, D. Gruss, M. Lipp, M. Minkin, D. Moghimi, F. Piessens, M. Schwarz, B. Sunar, J. Van Bulck, and Y. Yarom.
Fallout: Leaking Data on Meltdown-resistant CPUs.
In *Proceedings of the 26th ACM Conference on Computer and Communications Security (CCS)*, 2019.

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.
In *Proceedings of the 28th USENIX Security Symposium*, 2019.

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*, 2018.

J. T. Mühlberg and J. Van Bulck.
Reflections on post-Meltdown trusted computing: A case for open security processors.
;login: the USENIX magazine, Vol. 43(No. 3), Fall 2018.

M. Schwarz, M. Lipp, D. Moghimi, J. Van Bulck, J. Stecklina, T. Prescher, and D. Gruss.
ZombieLoad: Cross-Privilege-Boundary Data Sampling.
In *Proceedings of the 26th ACM Conference on Computer and Communications Security (CCS)*, 2019.

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

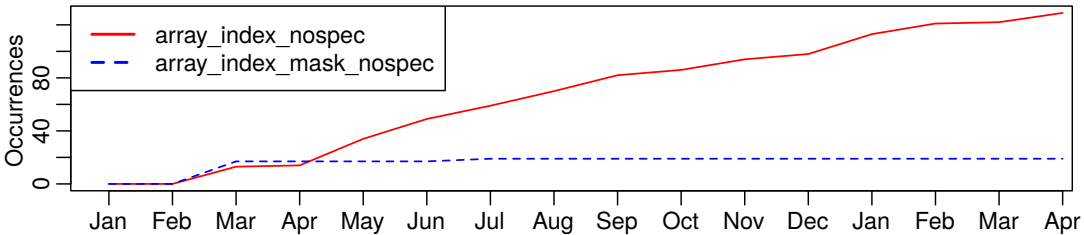
J. Van Bulck, F. Piessens, and R. Strackx.
Nemesis: Studying microarchitectural timing leaks in rudimentary CPU interrupt logic.
In *Proceedings of the 25th ACM Conference on Computer and Communications Security (CCS'18)*. ACM, October 2018.

S. van Schaik, A. Milburn, S. sterlund, P. Frigo, G. Maisuradze, K. Razavi, H. Bos, and C. Giuffrida.
RIDL: Rogue In-flight Data Load.
In *Proceedings of the 40th IEEE Symposium on Security and Privacy (S&P)*, May 2019.

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.

Spectre is here to stay: Evolution of Linux kernel gadget discovery

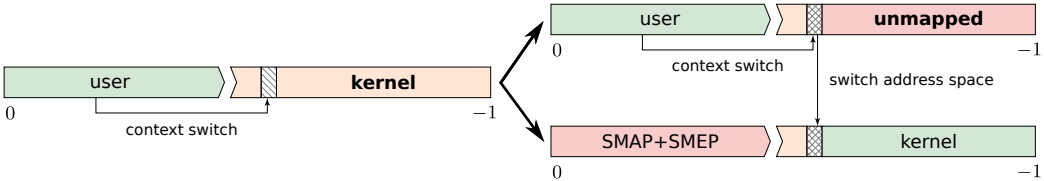


Canella et al. "A systematic evaluation of transient execution attacks and defenses", USENIX Security 2019

Mitigating Meltdown: Unmap kernel addresses from user space

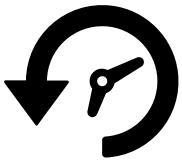
- OS software fix for faulty hardware ($\leftrightarrow$ future CPUs)
- Unmap kernel from user *virtual address space*

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



Gruss et al. "KASLR is dead: Long live KASLR", ESSoS 2017

Mitigating Foreshadow



1. Cache secrets in L1

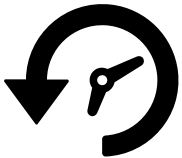
2. Unmap page table entry

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

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

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