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
Secure program: convert all input to *expected output*
Buffer overflow vulnerabilities: trigger unexpected behavior
A primer on software security

Safe languages & formal verification: preserve expected behavior

INPUT

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

DRAM memory
CPU cache timing side-channel

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

while true do
    maccess(&a);
endwhile

CPU + cache  DRAM memory
Cache hit: No DRAM access required for subsequent uses

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

CPU + cache

DRAM memory
Cache timing attacks in practice: Flush+Reload

```c
if secret do
    maccess(&a);
else
    maccess(&b);
endif
flush(&a);
start_timer
   maccess(&a);
end_timer
```

CPU + cache

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

DRAM memory
Cache timing attacks in practice: Flush+Reload

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

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

secret=1, load 'a' into cache

cache miss

CPU + cache

DRAM memory
Cache timing attacks in practice: Flush+Reload

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

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

CPU + cache DRAM memory

a

cache hit

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
**Side-channels:** observe *side-effects* of the computation
A primer on software security (revisited)

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

Key discrepancy:
- Programmers write sequential instructions

```c
int area(int h, int w)
{
    int triangle = (w*h)/2;
    int square = (w*w);
    return triangle + square;
}
```
Out-of-order and speculative execution

Key discrepancy:
- Programmers write **sequential** instructions
- Modern CPUs are inherently **parallel**

⇒ *Speculatively execute instructions ahead of time*

```c
int area(int h, int w)
{
    int triangle = (w*h)/2;
    int square = (w*w);
    return triangle + square;
}
```
Out-of-order and speculative execution

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 (!)

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

```
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

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

1. Cache secrets in L1
2. Unmap page table entry
3. Execute Meltdown
Foreshadow: Breaking the virtual memory abstraction

Arbitrary L1 cache read $\rightarrow$ 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)

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

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

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
```

Spectre v1: Speculative buffer over-read

Programmer intention: never access out-of-bounds memory

```java
if (idx < LEN)
{
    s = buffer[idx];
    t = lookup[s];
    ...
}
```
Spectre v1: Speculative buffer over-read

- 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**

```c
if (idx < LEN)
{
    s = buffer[idx];
    t = lookup[s];
    ...
}
```
Spectre v1: Speculative buffer over-read

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

```
LEN

user buffer  secret

if (idx < LEN)
{
    s = buffer[idx];
    t = lookup[s];
    ...
}
```
Mitigating Spectre v1: Inserting speculation barriers

Programmer intention: never access out-of-bounds memory

```c
if (idx < LEN)
{
    s = buffer[idx];
    t = lookup[s];
    ...
}
```
Mitigating Spectre v1: Inserting speculation barriers

- Programmer intention: never access out-of-bounds memory
- Insert **speculation barrier** to tell the CPU to halt the transient world until \( \text{idx} \) got evaluated \( \leftrightarrow \) performance 😞
Mitigating Spectre v1: Inserting speculation barriers

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...
<table>
<thead>
<tr>
<th>Age</th>
<th>Commit message (Expand)</th>
<th>Author</th>
<th>Files</th>
<th>Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 days</td>
<td>Merge git://git.kernel.org/pub/scm/linux/kernel/git/davem/net</td>
<td>Linus Torvalds</td>
<td>56</td>
<td>-274/+793</td>
</tr>
<tr>
<td>4 days</td>
<td>vhost: Fix Spectre v1 vulnerability</td>
<td>Jason Wang</td>
<td>1</td>
<td>0/+2</td>
</tr>
<tr>
<td>2018-10-19</td>
<td>Merge tag 'usb-4.19-final' of git://git.kernel.org/pub/scm/linux/kernel/git/g...</td>
<td>Greg Kroah-Hartman</td>
<td>7</td>
<td>-27/+65</td>
</tr>
<tr>
<td>2018-10-17</td>
<td>ptp: fix Spectre v1 vulnerability</td>
<td>Gustavo A. R. Silva</td>
<td>1</td>
<td>0/+6</td>
</tr>
<tr>
<td>2018-10-17</td>
<td>usb: gadget: storage: Fix Spectre v1 vulnerability</td>
<td>Gustavo A. R. Silva</td>
<td>1</td>
<td>0/+3</td>
</tr>
<tr>
<td>2018-10-16</td>
<td>RDMA/ucma: Fix Spectre v1 vulnerability</td>
<td>Gustavo A. R. Silva</td>
<td>1</td>
<td>0/+3</td>
</tr>
<tr>
<td>2018-10-16</td>
<td>IB/ucm: Fix Spectre v1 vulnerability</td>
<td>Gustavo A. R. Silva</td>
<td>1</td>
<td>0/+3</td>
</tr>
<tr>
<td>2018-09-18</td>
<td>tty: vt_ioctl: fix potential Spectre v1</td>
<td>Gustavo A. R. Silva</td>
<td>1</td>
<td>0/+4</td>
</tr>
<tr>
<td>2018-09-14</td>
<td>Merge tag 'char-misc-4.19-rc4' of git://git.kernel.org/pub/scm/linux/kernel/g...</td>
<td>Linus Torvalds</td>
<td>10</td>
<td>-34/+73</td>
</tr>
<tr>
<td>2018-09-12</td>
<td>Merge tag 'pci-v4.19-fixes-1' of git://git.kernel.org/pub/scm/linux/kernel/gi...</td>
<td>Linus Torvalds</td>
<td>8</td>
<td>-25/+41</td>
</tr>
<tr>
<td>2018-09-12</td>
<td>misc: hmc6352: fix potential Spectre v1</td>
<td>Gustavo A. R. Silva</td>
<td>1</td>
<td>0/+2</td>
</tr>
<tr>
<td>2018-09-11</td>
<td>switchtec: Fix Spectre v1 vulnerability</td>
<td>Gustavo A. R. Silva</td>
<td>1</td>
<td>0/+4</td>
</tr>
<tr>
<td>2018-08-29</td>
<td>Merge tag 'hwmon-for-linus-v4.19-rc2' of git://git.kernel.org/pub/scm/linux/kernel/k...</td>
<td>Linus Torvalds</td>
<td>5</td>
<td>-12/+32</td>
</tr>
<tr>
<td>2018-08-26</td>
<td>hwmon: (nct6775) Fix potential Spectre v1</td>
<td>Gustavo A. R. Silva</td>
<td>1</td>
<td>0/+2</td>
</tr>
<tr>
<td>2018-08-17</td>
<td>Merge tag 'drm-next-2018-08-17' of git://anongit.freedesktop.org/drm/drm</td>
<td>Linus Torvalds</td>
<td>44</td>
<td>-156/+346</td>
</tr>
</tbody>
</table>
Conclusions and take-away

Hardware + software patches

Update your systems! (+ disable HyperThreading)
Conclusions and take-away

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


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” (∼ mirror)

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