Leaky Processors: Stealing Your Secrets with Foreshadow

Jo Van Bulck

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

OWASP BeNeLux-Days, November 30, 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*
Safe languages & formal verification: preserve expected behavior
A primer on software security

**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/

DO WE JUST SUCK AT... COMPUTERS?

YUP. ESPECIALLY SHARED ONES.
SHARING IS NOT CARING

SHARING IS LOSING YOUR STUFF TO OTHERS
CPU cache timing side-channel

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

while true do
    maccess(&a);
end while
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
CPU cache timing side-channel

Cache hit: No DRAM access required for subsequent uses

while true do
    maccess(&a);
endwh

while true do
    maccess(&a);
endwh

Cache hit

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

a

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

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
```

```
dRAM memory
```

"secret=1, load 'a' into cache"

"cache miss"
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 DRAM memory

\[ a \]

cache hit

fast access(&a) $\rightarrow$ secret=1
if secret do
    maccess(&a);
else
    maccess(&b);
endif
flush(&a);
start_timer
   maccess(&b);
end_timer

slow access(&b) $\rightarrow$ 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 $\rightarrow$ *commit* results to normal world 😊
- Fail $\rightarrow$ *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 $\rightarrow$ *commit* results to normal world 😊
- Fail $\rightarrow$ *discard* results, compute again in normal world 😞

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

- *Delayed exception handling*
- *Control flow prediction*
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 exception handling
- Control flow prediction
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:

- CPU access control bypass
- Speculative buffer overflow/ROP
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.

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.

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. movb (%rsi), %al
5. shl $0xc, %rax
6. movq (%rdi, %rax), %rdi
7. 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. }

oracle array

cache hit
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]
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?

https://wired.com and https://arstechnica.com
Enclaved execution attack surface: TCB reduction
Enclaved execution attack surface: TCB reduction

- Trust vs. Untrusted
- OS kernel
- Hypervisor
- TPM
- CPU
- Mem
- HDD

1. Trusted
2. Untrusted
Enclaved execution attack surface: TCB reduction

Intel SGX promise: hardware-level isolation and attestation
Enclaved execution attack surface: TCB reduction

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]
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 (!)
L1 cache design: Virtually-indexed, physically-tagged
Page fault: Early-out address translation
Foreshadow-NG: Breaking the virtual memory abstraction

CPU micro-architecture

L1D → Tag? → Pass to out-of-order

PT walk? → vadrs

padrs

Page fault

L1-Terminal Fault: match *unmapped physical address* (!)
Foreshadow-NG: Breaking the virtual memory abstraction

CPU micro-architecture

L1D → Tag? → Pass to out-of-order

vadrs → padrs → PT walk?

 fail → Page fault

 SGX?

 ok → Allow

 fail → Abort page

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)

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

Intel microcode updates

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

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

jo@groplius:--$ cat /proc/cpuinfo | egrep "meltdown|lltf" -ml
bugs : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass lltf

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

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

jo@groplius:--$
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.


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.

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.

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 Pengelly | September 18, 2017

Foreshadow fallout: Dismantling the SGX ecosystem

Remote attestation and secret provisioning
Challenge-response to prove *enclave identity*
CPU-level key derivation

Intel == trusted 3rd 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 😊
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 (!)
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)
Research challenges: Universal classification and evaluation

Conclusions and take-away

Hardware + software patches

Update your systems! (+ disable HyperThreading)

https://foreshadowattack.eu/
Conclusions and take-away

Hardware + software patches

⇒ Update your systems! (+ disable HyperThreading)

⇒ New class of transient execution attacks

⇒ Importance of fundamental side-channel research

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

https://foreshadowattack.eu/
A systematic evaluation of transient execution attacks and defenses.  

KASLR is dead: Long live KASLR.  

Spectre attacks: Exploiting speculative execution.  

Meltdown: Reading kernel memory from user space.  

Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution.  

J. Van Bulck, F. Piessens, and R. Strackx.  
Nemesis: Studying microarchitectural timing leaks in rudimentary CPU interrupt logic.  

Foreshadow-NG: Breaking the virtual memory abstraction with transient out-of-order execution.  
Spectre v1: Speculative buffer over-read

Programmer intention: never access out-of-bounds memory

```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*

```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
- Side-channels leak out-of-bounds secrets to the real world

```c
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 \( idx \) 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 <em>Spectre v1</em> 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 <em>Spectre v1</em> vulnerability</td>
<td>Gustavo A. R. Silva</td>
<td>2</td>
<td>0/+6</td>
</tr>
<tr>
<td>2018-10-17</td>
<td>usb: gadget: storage: Fix <em>Spectre v1</em> vulnerability</td>
<td>Gustavo A. R. Silva</td>
<td>1</td>
<td>0/+1</td>
</tr>
<tr>
<td>2018-10-16</td>
<td>RDMA/ucma: Fix <em>Spectre v1</em> vulnerability</td>
<td>Gustavo A. R. Silva</td>
<td>1</td>
<td>0/+1</td>
</tr>
<tr>
<td>2018-10-16</td>
<td>l8/ucma: Fix <em>Spectre v1</em> vulnerability</td>
<td>Gustavo A. R. Silva</td>
<td>1</td>
<td>0/+1</td>
</tr>
<tr>
<td>2018-09-25</td>
<td>Merge tag 'tty-4.19-rc6' of git://git.kernel.org/pub/scm/linux/kernel/git/g...</td>
<td>Greg Kroah-Hartman</td>
<td>6</td>
<td>-7/+30</td>
</tr>
<tr>
<td>2018-09-18</td>
<td>tty_vt_ioctl: fix potential <em>Spectre v1</em></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/git/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/git/g...</td>
<td>Gustavo A. R. Silva</td>
<td>8</td>
<td>-25/+41</td>
</tr>
<tr>
<td>2018-09-12</td>
<td>misc: hmc6352: fix potential <em>Spectre v1</em></td>
<td>Gustavo A. R. Silva</td>
<td>1</td>
<td>0/+2</td>
</tr>
<tr>
<td>2018-09-11</td>
<td>switchtec: Fix <em>Spectre v1</em> vulnerability</td>
<td>Gustavo A. R. Silva</td>
<td>1</td>
<td>0/+2</td>
</tr>
<tr>
<td>2018-08-29</td>
<td>hwmn: 'for-linus-v4.19-rc2' of git://git.kernel.org/pub/scm/linux/kernel/git/g...</td>
<td>Linus Torvalds</td>
<td>5</td>
<td>-12/+32</td>
</tr>
<tr>
<td>2018-08-26</td>
<td>hwmn: (nct6775) Fix potential <em>Spectre v1</em></td>
<td>Linus Torvalds</td>
<td>44</td>
<td>-156/+346</td>
</tr>
</tbody>
</table>