Microarchitectural Security

Daniel Gruss
February 20, 2019

Graz University of Technology
FANTASTIC TIMERS
AND WHERE TO FIND THEM
HIGH-RESOLUTION MICROARCHITECTURAL ATTACKS IN JAVASCRIPT
Americoin,

Americoin

God shed his blocks on thee!

Americoin, Americoin, God shed his blocks on thee
Stealing Bitcoins?
Stealing Bitcoins?

Daniel Gruss — Graz University of Technology
Stealing Bitcoins?

Daniel Gruss — Graz University of Technology
Stealing Bitcoins?

Daniel Gruss — Graz University of Technology
Stealing Bitcoins?
Application

Untrusted part

Create Enclave

Operating System
Application

Untrusted part

Create Enclave

Trusted part

Call Gate

Trusted Fnc.

Operating System
Application

Untrusted part

Create Enclave

Call Trusted Fnc.

Trusted part

Call Gate

Trusted Fnc.

Operating System
SGX

Application

Untrusted part

Create Enclave

Call Trusted Fnc.

Trusted part

Call Gate

Trusted Fnc.

Operating System

Daniel Gruss — Graz University of Technology
Application

Untrusted part

Create Enclave

Call Trusted Fnc.

Call Gate

Trusted part

Trusted Fnc.

Operating System

SGX

www.tugraz.at

Daniel Gruss — Graz University of Technology
SGX

Application

Untrusted part

Create Enclave

Call Trusted Fnc.

Trusted part

Call Gate

Trusted Fnc.

Return

Operating System

Daniel Gruss — Graz University of Technology
Application

Untrusted part

Create Enclave

Call Trusted Fnc.

Call Gate

Trusted part

Trusted Fnc.

Return

Operating System

SGX

Daniel Gruss — Graz University of Technology
SGX

Application

Untrusted part

Create Enclave

Call Trusted Fnc.

...  

Trusted part

Call Gate

Trusted Fnc.

Return

Operating System

Daniel Gruss — Graz University of Technology
SGX

Application

Untrusted part

Create Enclave

Call Trusted Fnc.

... 

Trusted part

Call Gate

Trusted Fnc.

Return

Operating System

Daniel Gruss — Graz University of Technology
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.

Daniel Gruss — Graz University of Technology
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...\(^1\)

Attacking a weak RSA implementation inside SGX

...processed with a simple moving average... ¹

Attacking a weak RSA implementation inside SGX

...allows to clearly see the bits of the exponent\(^1\)

YOU CAN'T DO THAT!

THAT'S AGAINST THE RULES!
WANT TO DISCUSS THREAT MODELS NOW?
Physical Side Channels

- Power consumption
- Electro-magnetic radiation
- Temperature
- Photonic emission
- Acoustic emissions

Physical access usually relevant, but code execution on device usually not relevant.
Physical Side Channels

- Power consumption
Physical Side Channels

- Power consumption
- Electro-magnetic radiation
Physical Side Channels

- Power consumption
- Electro-magnetic radiation
- Temperature
Physical Side Channels

- Power consumption
- Electro-magnetic radiation
- Temperature
- Photonic emission
Physical Side Channels

- Power consumption
- Electro-magnetic radiation
- Temperature
- Photonic emission
- Acoustic emissions
Physical Side Channels

- Power consumption
- Electro-magnetic radiation
- Temperature
- Photonic emission
- Acoustic emissions

→ Physical access usually relevant, but code execution on device usually not relevant
Microarchitectural Attacks

1996
Microarchitectural Attacks

1996

2004
Microarchitectural Attacks

1996

2004

2006
Microarchitectural Attacks

1996

2004

2006

2009
Microarchitectural Attacks

1996

2004

2006

2009

2011
Microarchitectural Attacks

1996

2004

2006

2009

2011

2013
Microarchitectural Attacks

1996

2004

2006

2009

2011

2013

Daniel Gruss — Graz University of Technology
Microarchitectural Attacks

1996

2004

2006

2009

2011

2013

2014

Daniel Gruss — Graz University of Technology
Microarchitectural Attacks
Microarchitectural Attacks
Microarchitectural Attacks

1996
2004
2006
2009
2011
2013
2014

Daniel Gruss — Graz University of Technology
Microarchitectural Attacks

1996

2004

2006

2009

2011

2013

2014

2015

Daniel Gruss — Graz University of Technology
2016
Microarchitectural Attacks

2016

2017
Differences and Similarities

- threat model
- temporal component
- observer effect (destructive measurements)
- spatial component
Usually no physical access

Local code

Co-located code

Different meanings of "remote"

1. Attacker controls code in browser sandbox (e.g., [Ore+15; GMM16])

2. Attacker cannot control any code on the system
• Usually no physical access
• Usually no physical access
• Local code
Usually no physical access
Local code
Co-located code
- Usually no physical access
- Local code
- Co-located code
- Different meanings of “remote”
• Usually no physical access
• Local code
• Co-located code
• Different meanings of “remote”
  1. Attacker controls code in browser sandbox (e.g., [Ore+15; GMM16])
Microarchitectural Attacks - Threat Model

- Usually no physical access
- Local code
- Co-located code
- Different meanings of “remote”
  1. Attacker controls code in browser sandbox (e.g., [Ore+15; GMM16])
  2. Attacker cannot control any code on the system
Truly remote attacks...

Just a few examples:

- Remote timing attacks on crypto ([Ber04; BB05] and many more)
- ThrowHammer and NetHammer
- NetSpectre
Truly remote attacks...

Just a few examples:

- Remote timing attacks on crypto ([Ber04; BB05] and many more)
Truly remote attacks...

Just a few examples:

- Remote timing attacks on crypto ([Ber04; BB05] and many more)
- ThrowHammer and NetHammer
Truly remote attacks...

Just a few examples:

- Remote timing attacks on crypto ([Ber04; BB05] and many more)
- ThrowHammer and NetHammer
- NetSpectre
TIMING IS EVERYTHING
printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);
CPU Cache

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

Cache miss

Request
CPU Cache

Cache miss

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

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

Request
Response
```c
printf("%d", i);
printf("%d", i);
```

**CPU Cache**

- **Cache miss**
- **Cache hit**

**Request**

**Response**
CPU Cache

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

Cache miss

No DRAM access, much faster

Cache hit

Request

Response
CPU Cache

DRAM access, slow

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

Cache miss

Cache hit

No DRAM access, much faster

Request
Response
Flush+Reload

ATTACKER

flush
access

Shared Memory

VICTIM

access
Flush+Reload

ATTACKER

flush

access

Shared Memory

cached

VICTIM

access

cached
Flush+Reload

ATTACKER

flush

access

Shared Memory

VICTIM

access

Daniel Gruss — Graz University of Technology
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
Temporal Component: Timestamps

Physical Side Channels

Microarchitectural Attacks often around nanoseconds sometimes much lower
Physical Side Channels

- theoretical maximum accuracy of $5.4 \cdot 10^{-44}$s
Physical Side Channels

- theoretical maximum accuracy of $5.4 \cdot 10^{-44}$s
- feasible today: $850 \cdot 10^{-21}$s
Physical Side Channels

- theoretical maximum accuracy of $5.4 \cdot 10^{-44}$s
- feasible today: $850 \cdot 10^{-21}$s

Microarchitectural Attacks
Physical Side Channels

- theoretical maximum accuracy of $5.4 \cdot 10^{-44}$s
- feasible today: $850 \cdot 10^{-21}$s

Microarchitectural Attacks

- often around nanoseconds
Physical Side Channels

- theoretical maximum accuracy of $5.4 \cdot 10^{-44}$s
- feasible today: $850 \cdot 10^{-21}$s

Microarchitectural Attacks

- often around nanoseconds
- sometimes much lower
Temporal Component: Sampling Rate

Physical Side Channels
Physical Side Channels

- in the range of multiple GHz
Temporal Component: Sampling Rate

Physical Side Channels

- in the range of multiple GHz

Microarchitectural Attacks
Physical Side Channels

- in the range of multiple GHz

Microarchitectural Attacks

- usually varying frequency (depending on the attack)
Temporal Component: Sampling Rate

Physical Side Channels
- in the range of multiple GHz

Microarchitectural Attacks
- usually varying frequency (depending on the attack)
- between a few ns (< 1 GHz) and multiple seconds (< 1 Hz) (or even worse)
Physical Side Channels

- in the range of multiple GHz

Microarchitectural Attacks

- usually varying frequency (depending on the attack)
- between a few ns (< 1 GHz) and multiple seconds (< 1 Hz) (or even worse)
- strongly dependent on the specific attack
Physical Side Channels

- in the range of multiple GHz

Microarchitectural Attacks

- usually varying frequency (depending on the attack)
- between a few ns (< 1 GHz) and multiple seconds (< 1 Hz) (or even worse)
- strongly dependent on the specific attack
  - device under test = measurement device
Physical Side Channels

- in the range of multiple GHz

Microarchitectural Attacks

- usually varying frequency (depending on the attack)
- between a few ns (< 1 GHz) and multiple seconds (< 1 Hz) (or even worse)
- strongly dependent on the specific attack
  - device under test = measurement device
  - observer effect
device under test = measurement device

- measuring time takes some time
- limits the resolution
- measuring cache hits/misses manipulates the cache state
- virtually all measurements are destructive
Measurement Noise

Flush+Reload has no noise except for:

- Race condition between attacker and victim (observer effect)
- Speculative execution
- Prefetching...

Typically > 99.99% precision and recall.
Flush+Reload has no noise except for:

- Race condition between attacker and victim (observer effect)
- Speculative execution
- Prefetching

Typically > 99.99% precision and recall
Flush+Reload has no noise except for:

- Race condition between attacker and victim (observer effect)
- Speculative execution
- Prefetching

Typically > 99.99% precision and recall.
Flush+Reload has no noise except for:

- Race condition between attacker and victim (observer effect)
Flush+Reload has no noise except for:

- Race condition between attacker and victim (observer effect)
- Speculative execution
Flush+Reload has no noise except for:

- Race condition between attacker and victim (observer effect)
- Speculative execution
- Prefetching
Flush+Reload has no noise except for:

- Race condition between attacker and victim (observer effect)
- Speculative execution
- Prefetching
- ...
Flush+Reload has no noise except for:

- Race condition between attacker and victim (observer effect)
- Speculative execution
- Prefetching
- ...

→ Typically > 99.99% precision and recall
Measuring Processor Operations
• Very short timings

• `rdtsc` instruction: “cycle-accurate” timestamps

  
  ```
  ...
  rdtsc
  function()
  rdtsc
  ...
  ```
What are we measuring?

- Do you measure what you think you measure?
- *Out-of-order* execution → what is really executed?

```
rdtsc function() [...]
[daniel.gruss@tugraz.at]
```
• use pseudo-serializing instruction `rdtscp` (recent CPUs)
• use pseudo-serializing instruction `rdtscp` (recent CPUs)
• and/or use serializing instructions like `cpuid`
Accurate Microarchitecture Timing

- use pseudo-serializing instruction rdtscp (recent CPUs)
- and/or use serializing instructions like cpuid
- and/or use fences like mfence
• use pseudo-serializing instruction \texttt{rdtscp} (recent CPUs)
• and/or use serializing instructions like \texttt{cpuid}
• and/or use fences like \texttt{mfence}

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

- Cache Hits

Access time [CPU cycles]

Number of accesses

- 10^1
- 10^2
- 10^3
- 10^4
- 10^5
- 10^6
- 10^7
Memory Access Latency

Access time [CPU cycles]

Number of accesses

Cache Hits
Cache Misses

Daniel Gruss — Graz University of Technology
Flush+Reload had beautifully nice timings, right?

Well... steps of 2-4 cycles

only 35-70 steps between hits and misses

On some devices only 1-2 steps!
Flush+Reload had beautifully nice timings, right?

Well... steps of 2-4 cycles

only 35-70 steps between hits and misses

On some devices only 1-2 steps!
Temporal Component

Flush+Reload had beautifully nice timings, right?
Well... steps of 2-4 cycles
only 35-70 steps between hits and misses
On some devices only 1-2 steps!

Daniel Gruss — Graz University of Technology
Flush+Reload had beautifully nice timings, right?
- Flush+Reload had beautifully nice timings, right?
- Well... steps of 2-4 cycles
Flush+Reload had beautifully nice timings, right?

Well... steps of 2-4 cycles
- only 35-70 steps between hits and misses
Flush+Reload had beautifully nice timings, right?
Well... steps of 2-4 cycles
  - only 35-70 steps between hits and misses
On some devices only 1-2 steps!
Timer

- We can build our own timer
- We can build our **own timer**
- Start a thread that continuously increments a global variable
We can build our own timer
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

\[
\text{rdtsc} \quad 3 \quad 1 \text{ Timestamp} = \text{rdtsc}();
\]
CPU cycles one increment takes

```
1 while (1) {
2    timestamp++;
3 }
```
Self-built Timer

CPU cycles one increment takes

<table>
<thead>
<tr>
<th>Command</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdtsc</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>4.7</td>
</tr>
</tbody>
</table>

Assembly

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

<table>
<thead>
<tr>
<th>rdtsc</th>
<th>C</th>
<th>Optimized</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><img src="3" alt="3 cycles" /></td>
<td><img src="4.7" alt="4.7 cycles" /></td>
</tr>
</tbody>
</table>

#### Assembly

1. `mov &timestamp, %rcx`
2. `incl (%rcx)`
3. `jmp 1b`
## Self-built Timer

### CPU cycles one increment takes

<table>
<thead>
<tr>
<th>Method</th>
<th>Cycles</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></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Time</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></td>
</tr>
</tbody>
</table>

### Assembly Code

1. `mov &timestamp, %rcx`
2. `inc %rax`
3. `mov %rax, (%rcx)`
4. `jmp 1b`
## CPU cycles one increment takes

<table>
<thead>
<tr>
<th>Method</th>
<th>Time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>rdtsc</code></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 1: inc %rax
3 mov %rax, (%rcx)
4 jmp 1b
```
Modern Processor Design

Frontend
- Branch Predictor
- μOP Cache
- Allocation Queue

Instruction Fetch & PreDecode
- Instruction Queue
- 4-Way Decode

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

Execution Engine
- Scheduler
- Execution Units
- Load data
- Store data

Instruction Queue
- Instruction Fetch & PreDecode
- μOPs
- Branch Predictor
- Branch

Memory Subsystem
- Load Buffer
- Store Buffer
- L1 Data Cache
- L2 Cache

CDB
- Reorder buffer
- Scheduler
- Execution Units
- Load data
- Store data
- L1 Instruction Cache
- ITLB

Daniel Gruss — Graz University of Technology
Microarchitectural Defenses

device under test = measurement device

software defenses are possible
e.g., make sure attacker can’t compute in parallel to victim

how would that work in the physical world?
Microarchitectural Defenses

device under test = measurement device

software defenses are possible

e.g., make sure attacker can't compute in parallel to victim

how would that work in the physical world?
Microarchitectural Defenses

Device under test = measurement device

Software defenses are possible

* e.g., make sure attacker can't compute in parallel to victim

How would that work in the physical world?
device under test = measurement device
Microarchitectural Defenses

- device under test = measurement device
  → software defenses are possible
Microarchitectural Defenses

- device under test = measurement device
  → software defenses are possible
  - e.g., make sure attacker can’t compute in parallel to victim
- device under test = measurement device
  - software defenses are possible
  - e.g., make sure attacker can’t compute in parallel to victim
  - how would that work in the physical world?
Spatial Component

- Physical: Different offsets on the chip
- Microarchitectural: Different microarchitectural elements
  - More significant: Huge virtual address space
  - 2^48 different virtual memory locations
  - The location is often (part of) the secret
Spatial Component

- Physical: Different offsets on the chip
- Microarchitectural: Different microarchitectural elements
- More significant: Huge virtual address space
  - 2^48 different virtual memory locations
  - The location is often (part of) the secret
Spatia Component

physical: different offsets on the chip

microarchitectural: different microarchitectural elements

more significant: huge virtual address space

2^{48} different virtual memory locations

the location is often (part of) the secret
- physical: different offsets on the chip
Spatial Component

- physical: different offsets on the chip
- microarchitectural:
Spatial Component

- physical: different offsets on the chip
- microarchitectural:
  - different microarchitectural elements
Spatial Component

- physical: different offsets on the chip
- microarchitectural:
  - different microarchitectural elements
  - more significant: huge virtual address space
Spatial Component

- physical: different offsets on the chip
- microarchitectural:
  - different microarchitectural elements
  - more significant: huge virtual address space
  - $2^{48}$ different virtual memory locations
Spatial Component

- physical: different offsets on the chip
- microarchitectural:
  - different microarchitectural elements
  - more significant: huge virtual address space
  - \(2^{48}\) different virtual memory locations
  - the location is often (part of) the secret
% sleep 2: ./spy 300 7f05140a4000-7f051417b000 r-xp 0x20000 08:02 26 8050
/usr/lib/x86_64-linux-gnu/gedit/libgedit.so

shark$ ./spy
Side-Channel Attacks and Fault Attacks?
Physical

- Side-channel attacks
Physical

- Side-channel attacks
- Fault attacks
Physical

- Side-channel attacks
- Fault attacks
- What about cold boot attacks?
Physical

- Side-channel attacks
- Fault attacks
- What about cold boot attacks?

Microarchitectural
Attack Categories

Physical
- Side-channel attacks
- Fault attacks
- What about cold boot attacks?

Microarchitectural
- Side-channel attacks
Physical
- Side-channel attacks
- Fault attacks
- What about cold boot attacks?

Microarchitectural
- Side-channel attacks
- Fault attacks
Attack Categories

Physical
- Side-channel attacks
- Fault attacks
- What about cold boot attacks?

Microarchitectural
- Side-channel attacks
- Fault attacks
- What about Meltdown/Spectre?
*(volatile char*) 0;
array[84 * 4096] = 0;
• Flush+Reload over all pages of the array

![Chart showing access time in cycles for different pages.](chart.png)
• 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
- 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
used with authorization from Silicon Graphics, Inc. However, the authors make no claim that Mesa is in any way a compatible replacement for OpenGL or associated with Silicon Graphics, Inc.

... This version of Mesa provides GLX and DRI capabilities: it is capable of both direct and indirect rendering. For direct rendering, it can use DRI modules from the libg
• Basic Meltdown code leads to a crash (segfault)
• Basic Meltdown code leads to a crash (segfault)
• How to prevent the crash?
- Basic Meltdown code leads to a crash (segfault)
- How to prevent the crash?

Fault Handling

Fault Suppression

Fault Prevention
Meltdown with Fault Suppression

- Intel TSX to suppress exceptions instead of signal handler

```c
if (xbegin() == XBEGINTARTED) {
    char secret = *(char*) 0xffffffff81a000e0;
    array[secret * 4096] = 0;
    xend();
}

for (size_t i = 0; i < 256; i++) {
    if (flush_and_reload(array + i * 4096) == CACHE_HIT) {
        printf("%c\n", i);
    }
}
```
Speculative execution to prevent exceptions

```c
int speculate = rand() % 2;
size_t address = (0xffffffff81a000e0 * speculate) +
    ((size_t)&zero * (1 - speculate));
if (!speculate) {
    char secret = *(char*) address;
    array[secret * 4096] = 0;
}
for (size_t i = 0; i < 256; i++) {
    if (flush_and_reload(array + i * 4096) == CACHE_HIT) {
        printf("%c\n", i);
    }
}
```
Foreshadow / Foreshadow-NG

Booting from ROM...
early console in extract_kernel
input_data: 0x00000000001e0a276
input_len: 0x00000000003d48f8
output: 0x0000000001000000
output_len: 0x00000000011bc258
kernel_total_size: 0x000000000dec000
booted via startup_32()
Physical KASLR using RDTSC...
Virtual KASLR using RDTSC...

Decompressing Linux... Parsing ELF... Performing relocations... done.
Booting the kernel.

L1 Terminal Fault

Run `reader <pfn> [<cache miss threshold>]` to leak hypervisor data from the L1
index = 0;

char* data = "textKEY";

if (index < 4)
    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)`

`LUT[data[index] * 4096]`

`else`

`Prediction`

`Speculate`

`0`
index = 0;

cchar* data = "textKEY";

if (index < 4)

LUT[data[index] * 4096]
index = 1;

char* data = "textKEY";

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

char* data = "textKEY";

if (index < 4)

LUT[data[index] * 4096]

else

0

Prediction
Speculate

```c
index = 1;

char* data = "textKEY";

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

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]  0

Prediction
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]

else

0
index = 2;

char* data = "textKEY";

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

char* data = "textKEY";

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

Prediction

Spectre-PHT (v1)

Spectre-STL (v4): Ignore sanitizing write access and use unsanitized old value instead

Daniel Gruss — Graz University of Technology
index = 3;

char* data = "textKEY";

if (index < 4)

then

LUT[data[index] * 4096]

else

0
index = 3;

char* data = "textKEY";

if (index < 4)
then

Speculate

LUT[data[index] * 4096]

else

Prediction

0
index = 3;

char* data = "textKEY";

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

char* data = "textKEY";

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

char* data = "textKEY";

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

Prediction
index = 4;

char* data = "textKEY";

if (index < 4)

Speculate
then
LUT[data[index] * 4096]

Prediction

else

0
index = 4;

char* data = "textKEY";

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

char* data = "textKEY";

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

char* data = "textKEY";

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

Prediction
index = 5;

char* data = "textKEY";

if (index < 4)
{
    Speculate

    LUT[data[index] * 4096]

    then

    Prediction

    else

    0
index = 5;

char* data = "textKEY";

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

Execute
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

Prediction
Speculate

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;

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

Spectre-STL (v4): Ignore sanitizing write access and use unsanitized old value instead
`Animal* a = bird;`
Spectre v2

```cpp
Animal* a = bird;
```

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

Speculate

fly()

swim() -> Prediction

swim()
Animal* a = bird;

a->move()

fly()

swim()

swim()

Prediction

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

Execute

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

a->move()

fly()

Prediction

swim()

swim()

0
Animal* a = bird;

a->move()

fly()  

Prediction

fly()

swim()

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

a->move()

Speculate

fly()

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

Prediction

fly()

swim()

0
Animal* a = bird;

a->move()

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()

swim()

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

Spectre v2

Daniel Gruss — Graz University of Technology
Animal* a = fish;

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

Execute

0

Prediction

a->move()

fly()
Animal* a = fish;

a->move()

fly()

swim()  swim()

Prediction

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

a->move()

fly()  swim()  swim()

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

Spectre-BTB (v2): mistrain BTB → mispredict indirect jump/call
Animal* a = fish;

a->move()

fly()
swim()
swim()

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

0

Spectre-BTB (v2): mistrain BTB → mispredict indirect jump/call

Spectre-RSB (v5): mistrain RSB → mispredict return
• v1.1: Speculatively write to memory locations
• v1.1: Speculatively write to memory locations
  ➔ Many more gadgets than previously anticipated

---

v1.1: Speculatively write to memory locations

→ Many more gadgets than previously anticipated

v1.2: Ignore writable bit

---

v1.1: Speculatively write to memory locations
→ Many more gadgets than previously anticipated
v1.2: Ignore writable bit
→ = Meltdown-RW

---


Daniel Gruss — Graz University of Technology
operation #n

flush pipeline on wrong prediction

prediction

operate #n+2

possibly architectural transient execution

time

predict CF/DF
Meltdown

operation #n

exception

data

data dependency

Meltdown

raise

data

data dependency

operation #n+2

transient execution

possibly architectural

time

Daniel Gruss — Graz University of Technology
Mistraining Location

out-of-place/
same-address-space

Congruent branch

Address collision

Victim branch

Attacker

Congruent branch

Address collision

Shadow branch

Shared Branch Prediction State

in-place/
same-address-space

out-of-place/
cross-address-space

in-place/
cross-address-space
Classification Tree

Spectre-type microarchitectural buffer

Transient cause?

Spectre-PHT

Spectre-BTB

Spectre-RSB

Spectre-STL [32]

Cross-address-space

Same-address-space

mistraining strategy

PHT-CA-IP ★

PHT-SA-IP [54, 52]

PHT-SA-OP ★

BTB-CA-IP [54, 18]

BTB-SA-IP ★

BTB-SA-OP [18]

RSB-CA-OP [56]

RSB-CA-OP [64, 56]

RSB-SA-IP [64]

RSB-SA-OP [64, 56]

Cross-address-space

Same-address-space

prediction

Meltdown-type

Meltdown-NM [86]

Meltdown-AC ★

Meltdown-DE ★

Meltdown-PF

Meltdown-UF ★

Meltdown-SS ★

Meltdown-BR

Meltdown-GP [10, 41]

Meltdown-US [61]

Meltdown-P [93, 96]

Meltdown-RW [52]

Meltdown-PK ★

Meltdown-XD ★

Meltdown-SM ★

Meltdown-MPX [44]

Meltdown-BND ★
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>InvisiSpec</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>Sloth Reduction</th>
<th>SSBD/SSBB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel</td>
<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></td>
<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></td>
<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></td>
<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>
<tr>
<td>ARM</td>
<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></td>
<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></td>
<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></td>
<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>
<tr>
<td>AMD</td>
<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></td>
<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></td>
<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></td>
<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>real-world workload servers</td>
<td></td>
</tr>
<tr>
<td>Retpoline</td>
<td>5–10%</td>
<td></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>SYSmark®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>

Daniel Gruss — Graz University of Technology
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
- dedicate more time into identifying problems and not solely in mitigating known problems
Microarchitectural Security

Daniel Gruss
February 20, 2019
Graz University of Technology
References


