Software-based Microarchitectural Attacks

Daniel Gruss
January 23, 2019
Graz University of Technology

Frame Rate of Slide Deck: 11 fps
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
printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);

Cache miss

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

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

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

Daniel Gruss — Graz University of Technology
CPU Cache

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

Cache miss
Request
Response
Cache hit

Daniel Gruss — Graz University of Technology
CPU Cache

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

DRAM access, slow

Cache miss

Cache hit

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

drum access, slow

printf("%d", i);

Cache miss

Request
Response

i

Cache hit

No DRAM access, much faster

printf("%d", i);
Flush+Reload

ATTACKER

Shared Memory

VICTIM

flush
access

flush
access
Flush+Reload

ATTACKER

flush
access

Shared Memory

VICTIM

access

Daniel Gruss — Graz University of Technology
Flush+Reload

ATTACKER

flush

class

Shared Memory

VICTIM

access
Flush+Reload

ATTACKER

Shared Memory

VICTIM

flush
access
Flush+Reload

ATTACKER

Shared Memory

VICTIM

flush
access
access
Flush+Reload

ATTACKER
flush
access

Shared Memory

VICTIM
access

fast if victim accessed data, slow otherwise

Daniel Gruss — Graz University of Technology
Memory Access Latency

![Cache Hits](chart)

Access time [CPU cycles]

Number of accesses

Daniel Gruss — Graz University of Technology
Memory Access Latency

![Graph showing Memory Access Latency](image)

- **Cache Hits**
- **Cache Misses**

Access time [CPU cycles] vs. Number of accesses.
Cache Template Attack Demo

% sleep 2; ./spy 300 7f05140a4000-7f051417b000 r-xp 0x20000 08:02 26 8050
/usr/lib/x86_64-linux-gnu/gedit/libgedit.so

shark$ ./spy
HELLO FROM THE OTHER SIDE (DEMO):
VIDEO STREAMING OVER CACHE COVERT CHANNEL
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);
char data = *(char*)0xffffffff81a000e0;
printf("\%c\n", data);
char data = *(char*)0xfffffffff81a000e0;
printf("%c\n", data);

segfault at ffffffff81a000e0 ip
0000000000400535
sp 00007ffce4a80610 error 5 in reader
Adapted code

*(volatile char*)0;
array[84 * 4096] = 0; // unreachable
Flush+Reload over all pages of the array

Access time [cycles]

Page
Flush+Reload over all pages of the array

This also works on AMD and ARM!
• Combine the two things

```c
char data = *(char*)0xffffffff81a000e0;
array[data * 4096] = 0;
```
Flush+Reload again...

... Meltdown actually works.
attacker@meltdown ~/exploit %

victim@meltdown ~ %
• Kernel addresses in user space are a problem (the wall does not work)
• Kernel addresses in user space are a problem (the wall does not work)
• Why don’t we take the kernel addresses...
...and remove them if not needed?
...and remove them if not needed?
• User accessible check in hardware is not reliable
Kernel Address Isolation to have Side channels Efficiently Removed
Kernel Address Isolation to have Side channels Efficiently Removed
Let us get rid of bottlenecks
Use the naughty/nice list of last year
Finally, check predictions with list of this year
Throwing away wrongly manufactured presents
Correct predictions result in free time
SPECTRE
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
    0
LUT

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
{
    0
}
index = 1;

char* data = "textKEY";

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

Prediction
index = 1;

char* data = "textKEY";

if (index < 4)

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

char* data = "textKEY";

if (index < 4)

LUT[data[index] * 4096]

Speculate

then

Prediction

else

0
index = 1;

char* data = "textKEY";

if (index < 4)

then

LUT[data[index] * 4096]

else

Prediction

0
index = 2;

char* data = "textKEY";

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

0
index = 2;

char* data = "textKEY";

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

0
index = 2;

char* data = "textKEY";

if (index < 4)
index = 2;

char* data = "textKEY";

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

char* data = "textKEY";

if (index < 4)

Speculate

then

LUT[data[index] * 4096]

daniel gruss | graz university of technology
Spectre-PHT (aka Spectre Variant 1)

index = 3;

char* data = "textKEY";

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

Prediction

0
index = 4;

char* data = "textKEY";

if (index < 4)
then

LUT[data[index] * 4096]

else

0
index = 4;

char* data = "textKEY";

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

0
Spectre-PHT (aka Spectre Variant 1)

```c
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

Prediction

0
index = 5;

char* data = "textKEY";

if (index < 4)

LUT[data[index] * 4096] 0
Spectre-PHT (aka Spectre Variant 1)

index = 5;

char* data = "textKEY";

if (index < 4)

LUT[data[index] * 4096]

else

0

Speculate

then

Prediction

Daniel Gruss — Graz University of Technology
index = 5;
char* data = "textKEY";

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

Execute
0
index = 6;

char* data = "textKEY";

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

LUT[0]
```c
index = 6;

char* data = "textKEY";

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

0
```
index = 6;

char* data = "textKEY";

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

else
index = 6;

char* data = "textKEY";

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

Transient cause?

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

Meltdown-type
- Meltdown-NM
- Meltdown-AC
- Meltdown-DE
- Meltdown-PF
- Meltdown-PD
- Meltdown-PD
- Meltdown-RW
- Meltdown-US
- Meltdown-P
- Meltdown-BR
- Meltdown-GP

Microarchitectural buffer

In-place (IP) vs. out-of-place (OP)

Prediction

Fault type

Cross-address-space
- PHT-CA-IP
- PHT-CA-OP
- PHT-SA-IP
- PHT-SA-OP
- BTB-CA-IP
- BTB-CA-OP
- BTB-SA-IP
- BTB-SA-OP
- RSB-CA-IP
- RSB-CA-OP
- RSB-SA-IP
- RSB-SA-OP

Same-address-space
BLOCKCHAIN
<table>
<thead>
<tr>
<th>Attack</th>
<th>Defense</th>
<th>Intel</th>
<th>ARM</th>
<th>AMD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Invis/Spec</td>
<td>□ □ □ □</td>
<td>□ □ □ □</td>
<td>□ □ □ □</td>
</tr>
<tr>
<td></td>
<td>Safespec</td>
<td>● ● ● ●</td>
<td>● ● ● ●</td>
<td>● ● ● ●</td>
</tr>
<tr>
<td></td>
<td>DAVG</td>
<td>○ ○ ○ ○</td>
<td>○ ○ ○ ○</td>
<td>○ ○ ○ ○</td>
</tr>
<tr>
<td></td>
<td>RSB</td>
<td>● ○ ○ ○</td>
<td>● ○ ○ ○</td>
<td>● ○ ○ ○</td>
</tr>
<tr>
<td></td>
<td>Stuffing</td>
<td>○ ○ ● ○</td>
<td>○ ○ ● ○</td>
<td>○ ○ ● ○</td>
</tr>
<tr>
<td></td>
<td>Poison</td>
<td>○ ○ ● ○</td>
<td>○ ○ ● ○</td>
<td>○ ○ ● ○</td>
</tr>
<tr>
<td></td>
<td>Value Masking</td>
<td>○ ○ ○ ○</td>
<td>○ ○ ○ ○</td>
<td>○ ○ ○ ○</td>
</tr>
<tr>
<td></td>
<td>Index Isolation</td>
<td>○ ○ ○ ○</td>
<td>○ ○ ○ ○</td>
<td>○ ○ ○ ○</td>
</tr>
<tr>
<td></td>
<td>Site Isolation</td>
<td>○ ○ ○ ○</td>
<td>○ ○ ○ ○</td>
<td>○ ○ ○ ○</td>
</tr>
<tr>
<td></td>
<td>SLH</td>
<td>○ ○ ○ ○</td>
<td>○ ○ ○ ○</td>
<td>○ ○ ○ ○</td>
</tr>
<tr>
<td></td>
<td>YSNB</td>
<td>○ ○ ○ ○</td>
<td>○ ○ ○ ○</td>
<td>○ ○ ○ ○</td>
</tr>
<tr>
<td></td>
<td>IBRS</td>
<td>○ ○ ○ ○</td>
<td>○ ○ ○ ○</td>
<td>○ ○ ○ ○</td>
</tr>
<tr>
<td></td>
<td>STIPB</td>
<td>○ ○ ○ ○</td>
<td>○ ○ ○ ○</td>
<td>○ ○ ○ ○</td>
</tr>
<tr>
<td></td>
<td>IBPB</td>
<td>○ ○ ○ ○</td>
<td>○ ○ ○ ○</td>
<td>○ ○ ○ ○</td>
</tr>
<tr>
<td></td>
<td>Serialization</td>
<td>○ ○ ○ ○</td>
<td>○ ○ ○ ○</td>
<td>○ ○ ○ ○</td>
</tr>
<tr>
<td></td>
<td>Taint Tracking</td>
<td>○ ○ ○ ○</td>
<td>○ ○ ○ ○</td>
<td>○ ○ ○ ○</td>
</tr>
<tr>
<td></td>
<td>Timer Reduction</td>
<td>○ ○ ○ ○</td>
<td>○ ○ ○ ○</td>
<td>○ ○ ○ ○</td>
</tr>
<tr>
<td></td>
<td>Sloth Reduction</td>
<td>○ ○ ○ ○</td>
<td>○ ○ ○ ○</td>
<td>○ ○ ○ ○</td>
</tr>
<tr>
<td></td>
<td>SSBD/SSBB</td>
<td>○ ○ ○ ○</td>
<td>○ ○ ○ ○</td>
<td>○ ○ ○ ○</td>
</tr>
</tbody>
</table>

Mitigated (●), partially mitigated (○), not mitigated (□), theoretically mitigated (■), theoretically impeded (■), not theoretically impeded (□), or out of scope (◇).
Conclusions

- new class of attacks
Conclusions

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

- new class of 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
Software-based Microarchitectural Attacks

Daniel Gruss
January 23, 2019

Graz University of Technology