The impact of Meltre and Specdown on microkernel systems

Matthias Lange, Kernkonzept GmbH, FOSDEM 2019
“We need to talk about Meltre and Specdown.”

–Conf call with customer, early 2018
The impact of Meltdown and Spectre on the L4Re microkernel system
Questions

• Where we prepared?
• Did microkernel design principles protect or help us?
• What’s the impact of implemented mitigations?
Questions - Spoiler

• Where we prepared?  No

• Did microkernel design principles protected or helped us?  A little bit

• What’s the impact of implemented mitigations? 😥
Meltdown & Spectre
Set of vulnerabilities in modern CPUs
Meltdown
Classic virtual address space layout

Kernel

User
Classic virtual address space layout

4 GB

Kernel

3 GB

1:1

User
L4Re’s virtual address space layout

• Fiasco reserves fixed amount of memory for itself
  • Not all physical memory is mapped in the kernel
  • Uses big pages for mapping
  • Mapping may include user memory
L4Re’s virtual address space layout
Solution: Kernel address space

• Move kernel into its own address space
  • Fiasco uses a CPU local address space

• User address space only maps absolutely necessary parts
  • GDT, TSS, entry / exit stack, UTCBs
Benchmarks - PTI
Benchmarks - Meta

• Baseline
  • Fiasco GitHub commit 566cc120, January 1st, 2018

• Head
  • Fiasco GitHub commit 591c8c0b, January 7th, 2019

• Compiler: kernel clang 6, userland gcc 7.3

• Core i7-5700EQ, 2.60GHz

• Contact me if interested in raw data
Benchmarks - Scenario 1
Benchmarks - Scenario 2
Micro benchmarks - pingpong, PTI

- IPC inter AS: 1.561 (Baseline 2018), 3.371 (PTI)
- Context switch: 1.759 (Baseline 2018), 2.586 (PTI)
- Thread switch (intra): 422 (Baseline 2018), 963 (PTI)
Benchmarks - Scenario 1, PTI

Baseline 2018: 9,37 Gbit/s
PTI: 9,27 Gbit/s
Benchmarks - Scenario 2, PTI

- Baseline 2018: 5.14 Gbit/s
- PTI: 3.17 Gbit/s
Spectre
Spectre

- Indirect branch prediction speculatively access data causing side effects

```java
if (idx < array1_size) {
    value = array2[array1[idx] * 256];
}
```
Spectre NG

- Speculative access to FPU state while current context is not the owner
- Fiasco uses lazy FPU switching
Spectre NG - Mitigation

• Fiasco now supports eager switching on x86

• Does this incur any performance loss?
Benchmarks - Eager FPU switching
Micro benchmarks - pingpong, PTI, eager FPU

<table>
<thead>
<tr>
<th>Baseline 2018</th>
<th>PTI</th>
<th>PTI, eager FPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPC inter AS</td>
<td>1.561</td>
<td>3.371</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.729</td>
</tr>
<tr>
<td>Context switch</td>
<td>1.759</td>
<td>2.586</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.918</td>
</tr>
<tr>
<td>Thread switch (intra)</td>
<td>422</td>
<td>963</td>
</tr>
</tbody>
</table>
Benchmarks - Scenario 1, PTI, eager FPU

Baseline 2018: 9,37Gbit/s
PTI: 9,27Gbit/s
PTI, eager FPU: 9Gbit/s
Benchmarks - Scenario 2, PTI, eager FPU

- **Baseline 2018**: 5.14 Gbit/s
- **PTI**: 3.17 Gbit/s
- **PTI, eager FPU**: 3.12 Gbit/s
Spectre continued

• Most variants do not work across process boundaries

• Usually code execution required
Spectre continued - Mitigations

• Fiasco mitigations
  • Indirect branch prediction barrier at kernel entry
  • Full prediction barrier at context switch
  • (microcode loading functionality)
Benchmarks - IBRS 😥
Micro benchmarks - pingpong, IBRS

- IPC inter AS: 1.561 (Baseline 2018), 3.371 (PTI), 3.729 (PTI, eager FPU), 16.601 (PTI, IBRS, eager FPU)
- Context switch: 1.759 (Baseline 2018), 2.586 (PTI), 2.918 (PTI, eager FPU), 8.820 (PTI, IBRS, eager FPU)
- Thread switch (intra): 422 (Baseline 2018), 963 (PTI), 1149 (PTI, eager FPU), 2638 (PTI, IBRS, eager FPU)
Benchmarks - Scenario 1, IBRS

- **Baseline 2018**: 9.37 Gbit/s
- **PTI**: 9.27 Gbit/s
- **PTI, eager FPU**: 9 Gbit/s
- **PTI, IBRS, eager FPU**: 7.68 Gbit/s
Benchmarks - Scenario 2, IBRS

- Baseline 2018: 5.14 Gbit/s
- PTI: 3.17 Gbit/s
- PTI, eager FPU: 3.12 Gbit/s
- PTI, IBRS, eager FPU: 1.28 Gbit/s
Foreshadow

L1 Terminal Fault
L1 Terminal Fault

- Affects OS / SMM, VT-x and SGX
- SGX not supported in L4Re
  - Don’t care
- SMM needs to protect itself
L1 Terminal Fault - L4Re mitigations

- OS
  - Fiasco is not vulnerable
  - We zero our PTEs
- VT-x is nasty
  - Microcode update
    - New MSR and new instruction for L1D flush
  - Flush L1D on every vmresume
Benchmarks - Sorry, no benchmarks for L1TF.
But there is one more thing ...
One more thing

- All features / mitigations are configurable

- You can turn off
  - PTI
  - Eager FPU
  - IBRS

- How does this compare to the 2018 baseline?
Micro benchmarks - pingpong

Baseline 2018 | PTI | PTI, eager FPU | PTI, IBRS, eager FPU | Baseline 2019

IPC inter AS

Context switch

Thread switch (intra)
Micro benchmarks - pingpong

- IPC inter AS
  - Baseline 2018: 1.561
  - Baseline 2019: 1.422
  - PTI: 3.371
  - PTI, eager FPU: 3.729

- Context switch
  - Baseline 2018: 1.759
  - Baseline 2019: 1.733
  - PTI: 2.586
  - PTI, eager FPU: 2.918

- Thread switch (intra)
  - Baseline 2018: 422
  - Baseline 2019: 425
  - PTI: 963
  - PTI, eager FPU: 1.149
Benchmarks - Scenario 1

- Baseline 2018: 9.37 Gbit/s
- Baseline 2019: 9.29 Gbit/s
- PTI: 9.27 Gbit/s
- PTI, eager FPU: 9.0 Gbit/s
Benchmarks - Scenario 2

Baseline 2018
Baseline 2019
PTI
PTI, eager FPU

iperf3

5.14 Gbit/s
5.14 Gbit/s
3.17 Gbit/s
3.12 Gbit/s
Conclusion
“Fiasco is still not the fastest microkernel in the world.”

– Me
Conclusion

• Some bugs did not hit as hard
• “missing” features helped us
• Dramatic performance impact
  • Consider alternatives compared to microcode
• Reconsider existing legacy implementations
  • Removed IO page fault
• What to expect in the future? How can we proactively act?
• gcc vs. clang
THANK YOU