Microarchitectural Attacks on the Cloud

Cat-and-Mouse Games between Hypervisors and the Microarchitecture

Lukas Giner, Andreas Kogler, Sandro Letter

May 10, 2021

IAIK – Graz University of Technology

What is the purpose of page deduplication?

Page deduplication

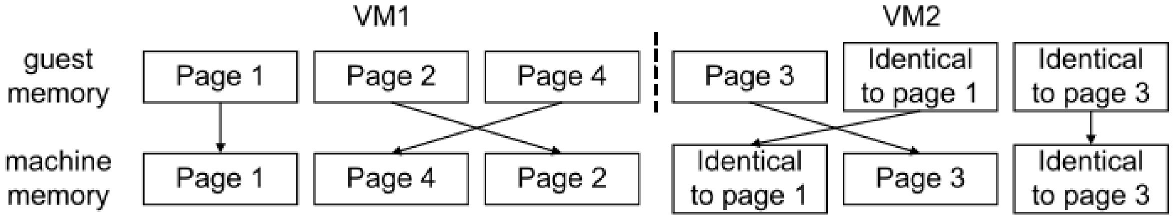
- Reduce amount of pages

Page deduplication

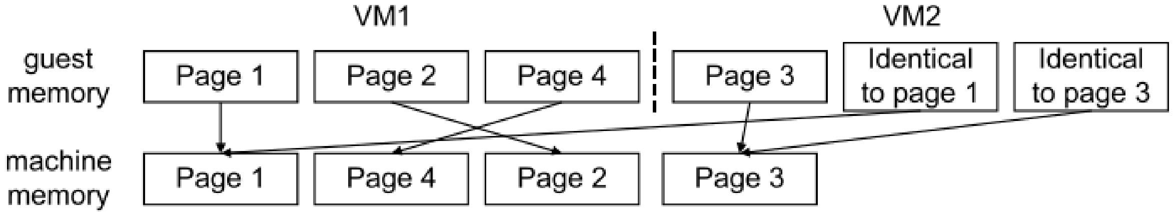
- Reduce amount of pages
- COW-principle

Page deduplication

- Reduce amount of pages
- COW-principle
- Efficient way of memory usage



(a) normal condition



(b) with memory de-duplication

Example: Attack to find a specific application in another VM

Page deduplication

- Fill pages with random data and data for deduplication

Page deduplication

- Fill pages with random data and data for deduplication
- Measuring of time to overwrite pages

Page deduplication

- Fill pages with random data and data for deduplication
- Measuring of time to overwrite pages
- Fill pages with application based information

Page deduplication

- Fill pages with random data and data for deduplication
- Measuring of time to overwrite pages
- Fill pages with application based information
- Wait for deduplication to take place

Page deduplication

- Fill pages with random data and data for deduplication
- Measuring of time to overwrite pages
- Fill pages with application based information
- Wait for deduplication to take place
- Modify pages and measure time to overwrite

So we now know another VM uses this application.

What to do next?

Page deduplication

- Attack it with flush + reload

Page deduplication

- Attack it with flush + reload
- Because we now have shared memory

Are there counter measures?

Page deduplication

- Read only pages

Page deduplication

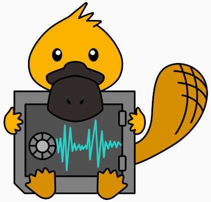
- Read only pages
- Obfuscation code

Page deduplication

- Read only pages
- Obfuscation code
- ? Memory sanitization ?

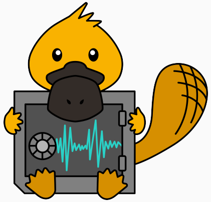
In the meanwhile, a Platypus appeared in the cloud!

Platypus



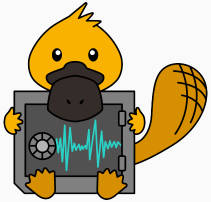
- Energy measurement interface

Platypus



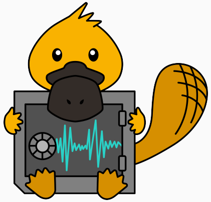
- Energy measurement interface
- Slow, but powerfull side channel

Platypus



- Energy measurement interface
- Slow, but powerfull side channel
- Xen did not restrict access to the interface

Platypus



- Energy measurement interface
- Slow, but powerfull side channel
- Xen did not restrict access to the interface

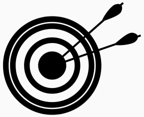
Takeaway

Carefully choose what the guests can access.

How does virtualization extend the attack surface?

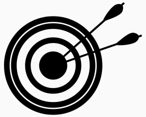
iTLB Multihit

- Instruction decoding accesses iTLB



iTLB Multihit

- Instruction decoding accesses iTLB
- Attacker fills iTLB with two mappings



iTLB Multihit

- Instruction decoding accesses iTLB
- Attacker fills iTLB with two mappings
  - Normal 4kB page
  - Huge page with different memory type

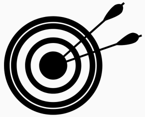
iTLB Multihit

- Instruction decoding accesses iTLB
- Attacker fills iTLB with two mappings
  - Normal 4kB page
  - Huge page with different memory type
- System Lockup, Denial of Service

iTLB Multihit

- Instruction decoding accesses iTLB
- Attacker fills iTLB with two mappings
  - Normal 4kB page
  - Huge page with different memory type
- System Lockup, Denial of Service
- On some CPUs fixed in software

iTLB Multihit



- Instruction decoding accesses iTLB
- Attacker fills iTLB with two mappings
  - Normal 4kB page
  - Huge page with different memory type
- System Lockup, Denial of Service
- On some CPUs fixed in software

Takeaway

The VM threat model allows for stronger attacks.

But what about Meltdown?

Meltdown Paravirtualization

- Xen PV Hypervisor

Meltdown Paravirtualization

- Xen PV Hypervisor
- Shares address space

Meltdown Paravirtualization

- Xen PV Hypervisor
- Shares address space
- Classical Meltdown attack leaks HV data

Meltdown Paravirtualization

- Xen PV Hypervisor
- Shares address space
- Classical Meltdown attack leaks HV data
- Proposed solution: migrate to HVM

Meltdown Paravirtualization

- Xen PV Hypervisor
- Shares address space
- Classical Meltdown attack leaks HV data
- Proposed solution: migrate to HVM

Takeaway

Share as little state as possible. Use HVM if applicable.

So, no shared mappings, are we save now?

Lazy FP

- FPU/SIMD context switches are expensive

Lazy FP

- FPU/SIMD context switches are expensive
- Switch when access triggers #NM exception

Lazy FP

- FPU/SIMD context switches are expensive
- Switch when access triggers #NM exception
- Suppress fault and encode data in side channel

Lazy FP

- FPU/SIMD context switches are expensive
- Switch when access triggers #NM exception
- Suppress fault and encode data in side channel
- Leak e.g. AES-NI registers

Lazy FP

- FPU/SIMD context switches are expensive
- Switch when access triggers #NM exception
- Suppress fault and encode data in side channel
- Leak e.g. AES-NI registers
- Disabling Lazy FP switches

Lazy FP

- FPU/SIMD context switches are expensive
- Switch when access triggers #NM exception
- Suppress fault and encode data in side channel
- Leak e.g. AES-NI registers
- Disabling Lazy FP switches

Takeaway

Lazy mechanics often introduce security risks.

So, don’t share mappings and don’t be lazy! Are we done now?

Foreshadow-VMM

Foreshadow-VMM

- Foreshadow by Jo van Bulck et.al + Ofir Weisse et al.

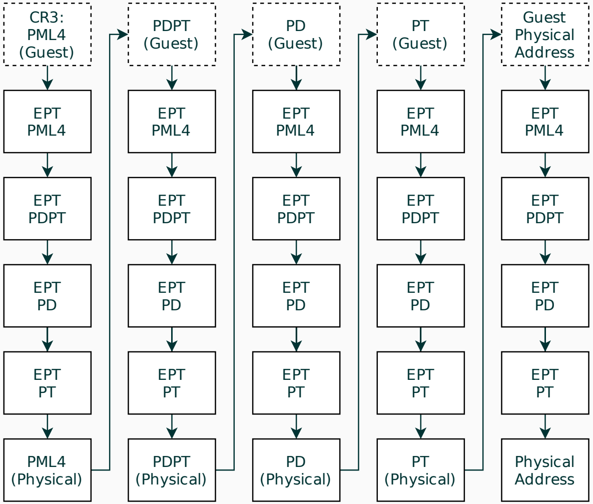
Foreshadow-VMM

- Foreshadow by Jo van Bulck et.al + Ofir Weisse et al.
- aka L1TF aka Meltdown-P

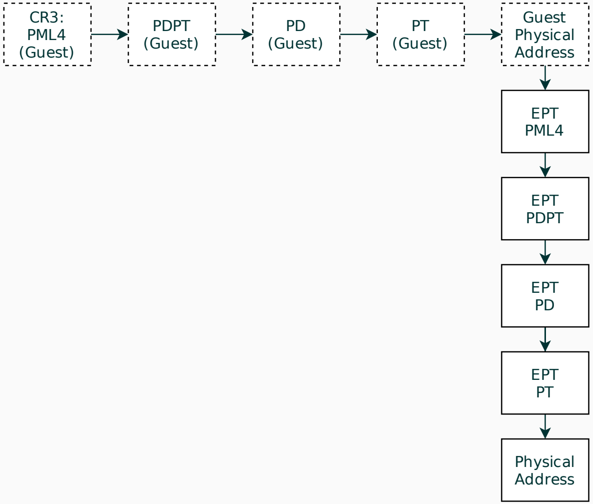
Foreshadow-VMM

- Foreshadow by Jo van Bulck et.al + Ofir Weisse et al.
- aka L1TF aka Meltdown-P
- (ab)uses the present bit in PTE

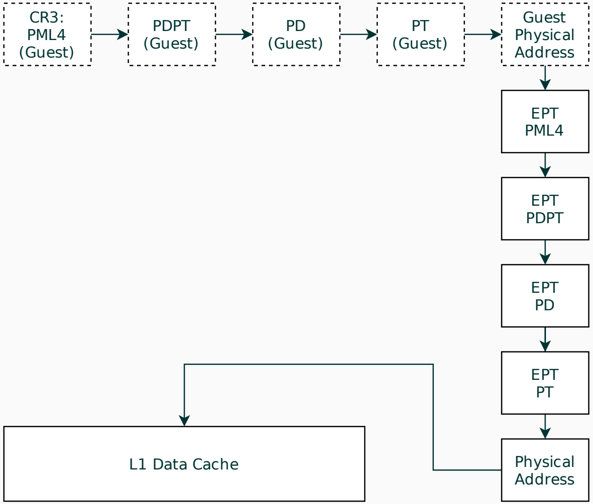
Foreshadow-VMM EPT - where things go wrong



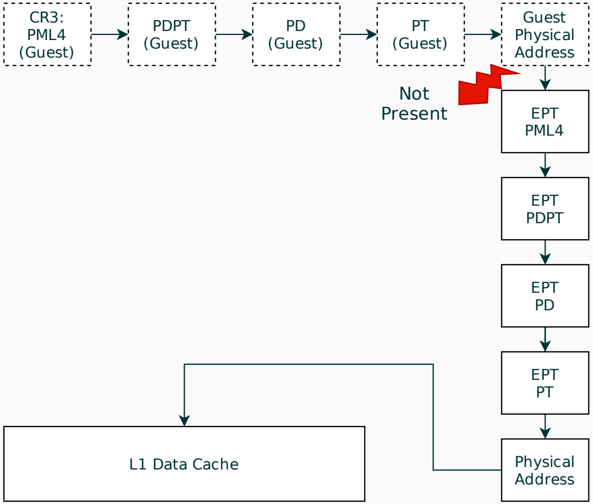
Foreshadow-VMM EPT - where things go wrong



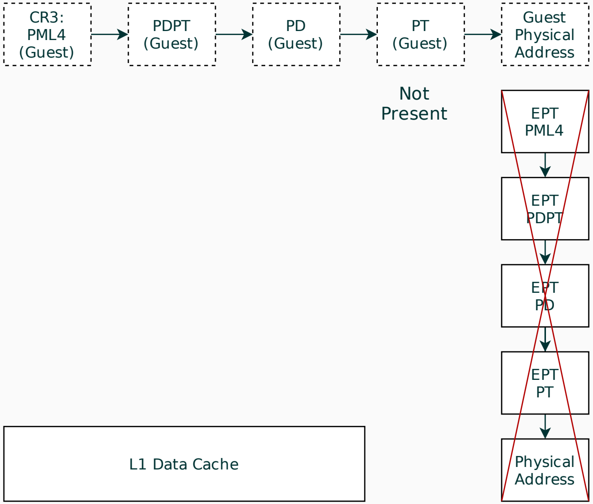
Foreshadow-VMM EPT - where things go wrong



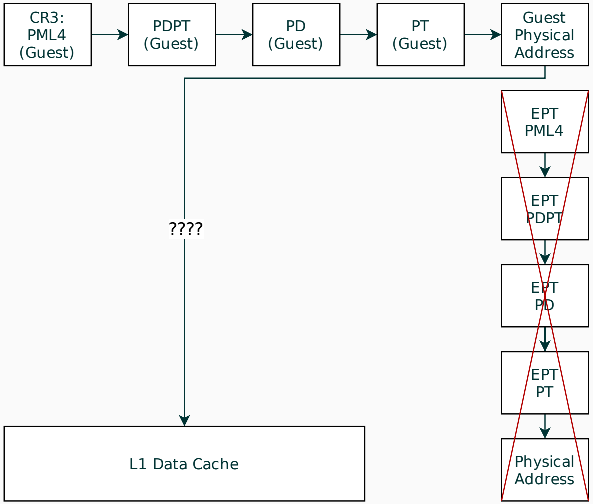
Foreshadow-VMM EPT - where things go wrong



Foreshadow-VMM EPT - where things go wrong



Foreshadow-VMM EPT - where things go wrong



Foreshadow-VMM - Exploiting

1. VM creates mapping, present $= 0$

Foreshadow-VMM - Exploiting

1. VM creates mapping, present $= 0$
2. CPU stops translation before second level, what now?

Foreshadow-VMM - Exploiting

1. VM creates mapping, present $= 0$
2. CPU stops translation before second level, what now?
3. Encode the data from L1 into the cache!

Foreshadow-VMM - Exploiting

1. VM creates mapping, present = 0
2. CPU stops translation before second level, what now?
3. Encode the data from L1 into the cache!
4. Retrieve with Flush+Reload

Foreshadow-VMM - Exploiting

1. VM creates mapping, present $= 0$
2. CPU stops translation before second level, what now?
3. Encode the data from L1 into the cache!
4. Retrieve with Flush+Reload
5. Celebrate

Foreshadow-VMM - Mitigation

- Fixed in hardware, but until then?

Foreshadow-VMM - Mitigation

- Fixed in hardware, but until then?
- Flush L1 Cache on entry $\rightarrow$ no more leakage

Foreshadow-VMM - Mitigation

- Fixed in hardware, but until then?
- Flush L1 Cache on entry $\rightarrow$ no more leakage.. right?

Foreshadow-VMM - Mitigation

- Fixed in hardware, but until then?
- Flush L1 Cache on entry → no more leakage.. right?
- Hyperthreading ruins the day.

Foreshadow-VMM - Mitigation

- Fixed in hardware, but until then?
- Flush L1 Cache on entry → no more leakage.. right?
- Hyperthreading ruins the day.
- Turn HT off, or only share cores with trusted VMs

Conclusion

- ISA says what should happen..

Conclusion

- ISA says what should happen.. μarch decides how

Conclusion

- ISA says what should happen.. μarch decides how
- VMM can fix μarch vulnerabilities..

Conclusion

- ISA says what should happen.. μarch decides how
- VMM can fix μarch vulnerabilities.. but also facilitate them!

Conclusion

- ISA says what should happen.. μarch decides how
- VMM can fix μarch vulnerabilities.. but also facilitate them!
- VMM has to consider both!