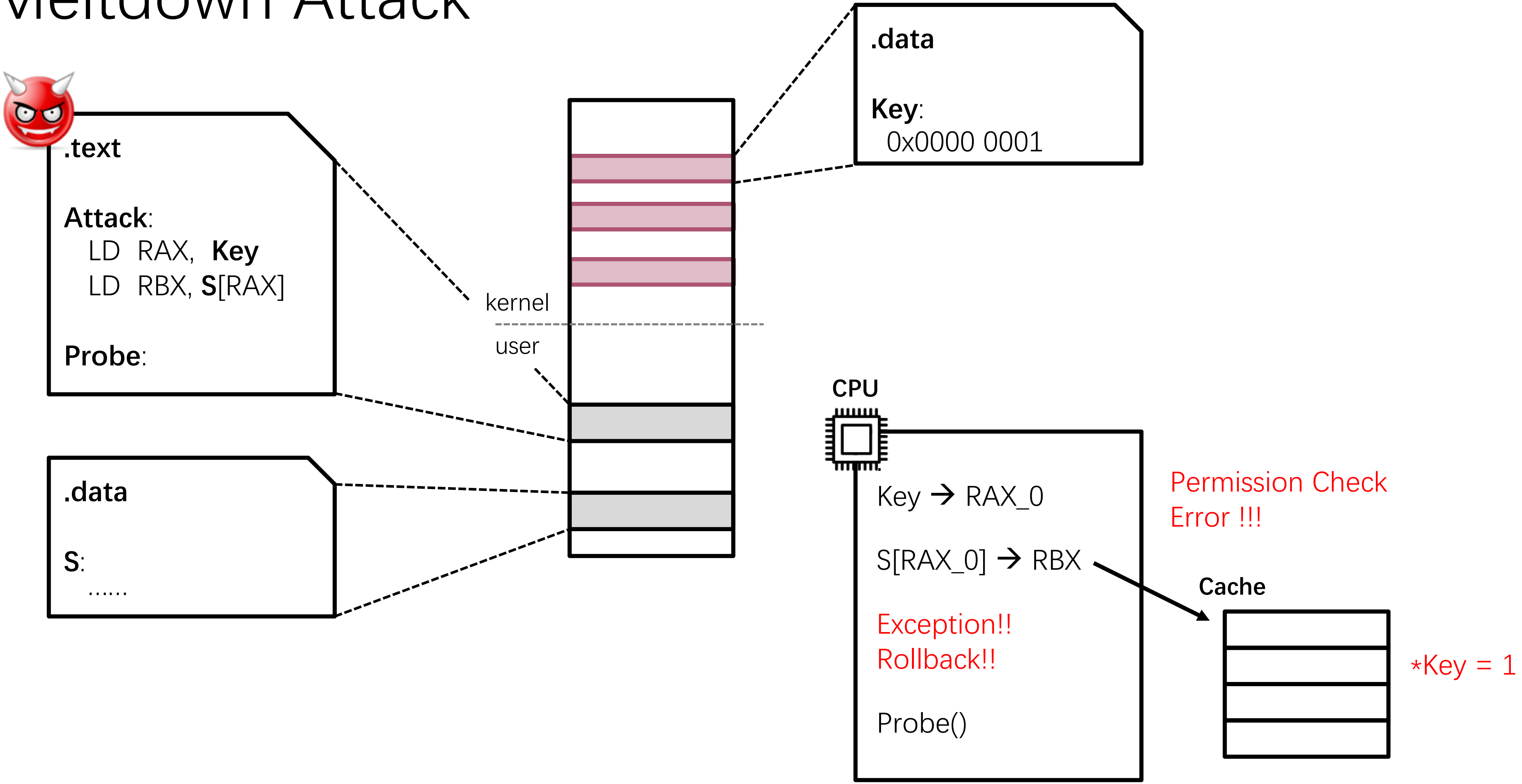
IPADS, Shanghai Jiao Tong University

上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

EPTI: Efficient Defence against Meltdown Attack for Unpatched VMs

Zhichao Hua, Dong Du, Yubin Xia, Haibo Chen, Binyu Zang

Meltdown Attack



KPTI (Kernel Page Table Isolation)

- Meltdown
  - Hardware bug at pipeline level
  - Exist in all Intel CPUs
  - Cannot fixed by micro-code patch
- KPTI
  - Two page tables (for kernel and user mode)
  - Remove kernel mapping in user page table
  - Switching page table during user/kernel switching

Problems of KPTI

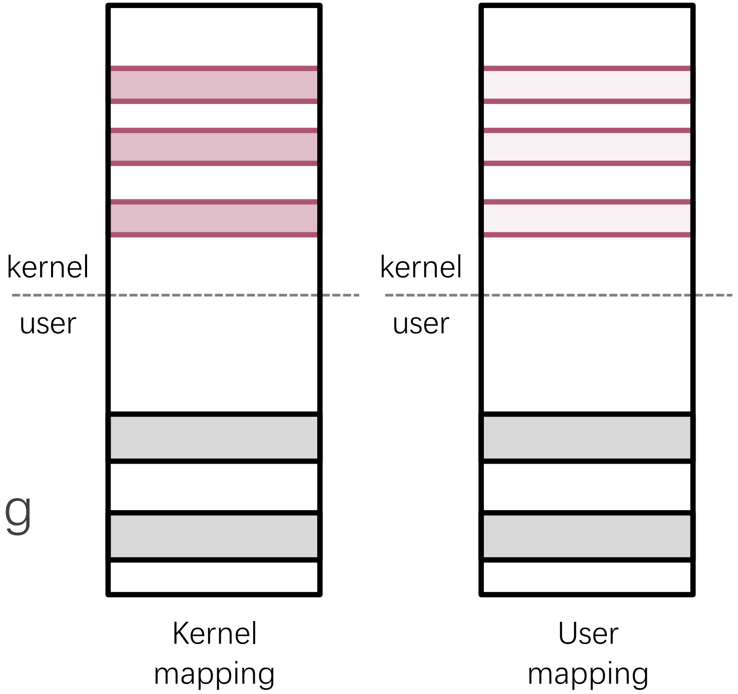
- KPTI has to be patched **manually**
  - In cloud environment, many cloud users are not capable of doing such system maintenance
- KPTI patch depends on **specific versions of kernel**
  - "just got the Meltdown update to kernel linux-image-4.4.0-108-generic but this does not boot at all"
- Incur non-trivial **performance slowdown**
  - Up to 30% overhead in VMs

Goals of EPTI

- Security
  - Defend **against Meltdown**
- Usability
  - Can be applied to **unpatched guest VMs** (independent on kernel version)
  - **Seamless deployment** without rebooting the VM
- Performance
  - **Lower** performance overhead than KPTI

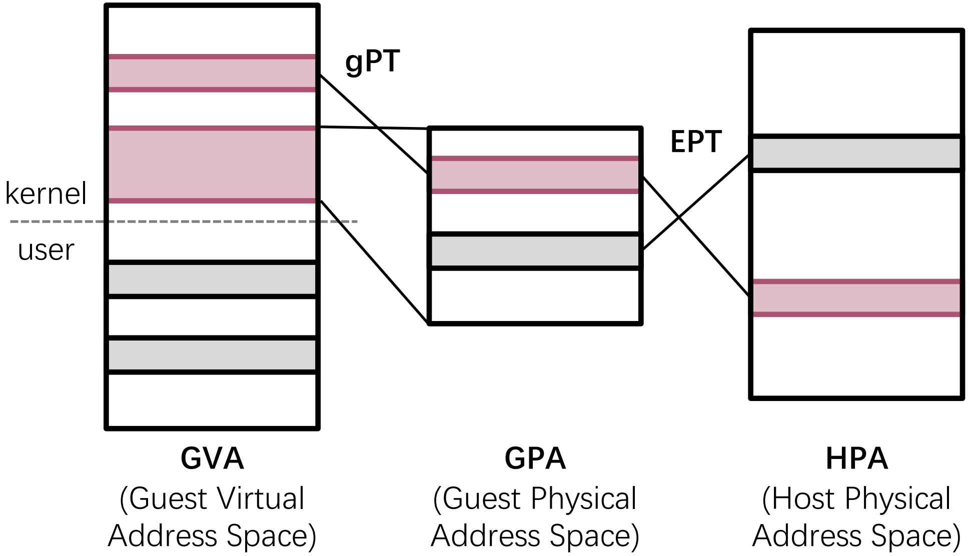
Overview

- Construct two different mappings
  - For guest user and kernel
  - By controlling EPT
    - EPT-k for kernel and EPT-u for user
- Enable protection on guest VM
  - Add trampoline at kernel enter/exit point
  - Leverage VMFUNC to perform EPT switching
  - Binary rewriting



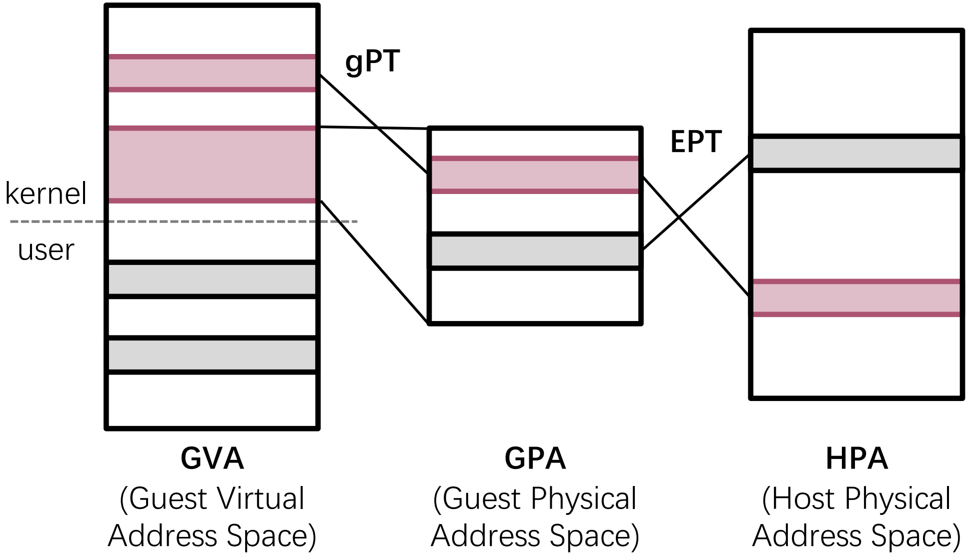
Kernel Space Isolation

- Naïve method:
  - Remove kernel GPA-to-HPA mapping
  - Difficult to identify kernel GPA
    - Kernel always map all GPA



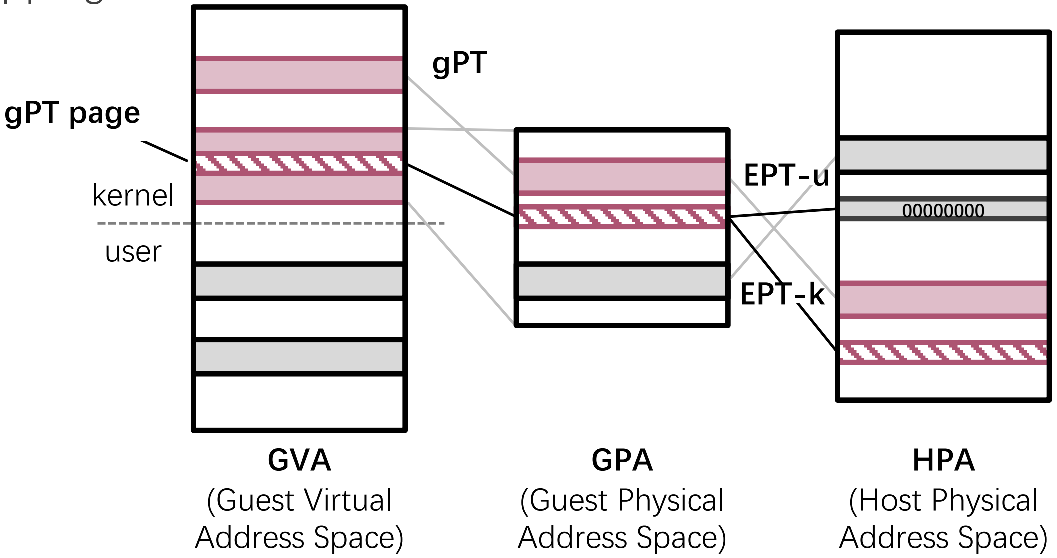
Kernel Space Isolation

- EPTI method:
  – Remove kernel GVA-to-GPA mapping



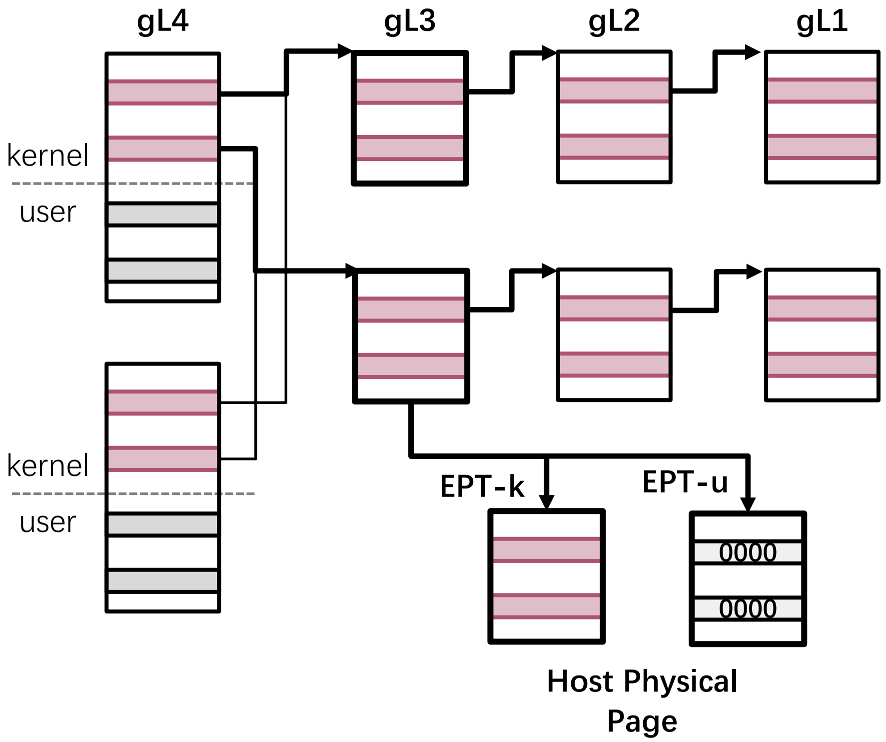
Kernel Space Isolation

- EPTI method:
  - Remove kernel GVA-to-GPA mapping
  - Remap **gPT page** for kernel mapping
    - Contains kernel GVA-to-GPA mapping
    - To a zeroed HPA page



Kernel Space Isolation

- Remap gL3 page
  - All processes share the same gL3 pages for kernel mapping
  - Remap gL3 pages to a new host physical pages in EPT-u
  - Zero the kernel GVA-to-GPA maping in EPT-u



Tracing gL3 pages

- Trace all enabled kernel gL3 pages
  - Step-1: Trap *MOV to CR3* to get all gL4 pages
  - Step-2: Trap all **write access** to gL4 pages to get enabled kernel gL3 page
- Problem: causes a lot of VMExits
  - Both loading CR3 and write gL4 pages cause VMExits
  - CPU updating access/dirty-bit causes VMExits

OPT-1. Selectively Tracking Guest CR3

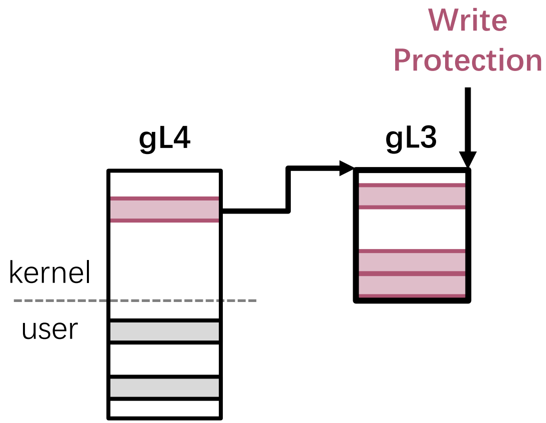
- Only need to trap loading **new** guest CR3
- Not trap loading **frequently-loaded old** guest CR3
  - Four *CR3_TARGET_VALUE* fields in VMCS
    - Load-CR3 with the value in these fields will **not** cause VMExit

OPT-2. Trapping gL3 Instead of gL4

- Kernel memory layout is fixed
  - Linux reserves memory regions for different usages
    - E.g., 0xffff880000000000 to 0xffffc7ffffffffff for direct map
    - E.g., ffffc90000000000 - ffffe8ffffffffff for vmalloc/ioremap
  - Only **parts** of these regions change at runtime
    - Kernel creates **a new gL3 page** (mapping 512GB) when **all entries** of existing gL3 pages are in use

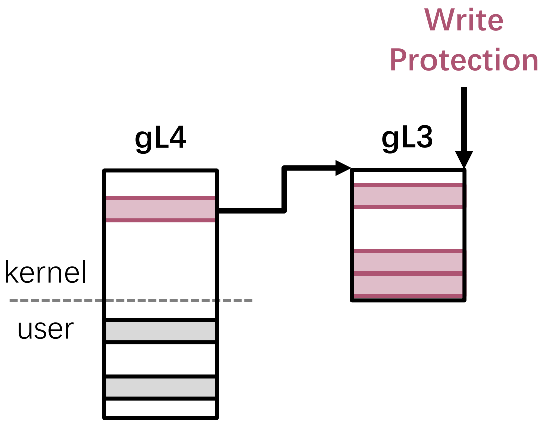
OPT-2. Trapping gL3 Instead of gL4

- Trap write access on kernel gL3 pages
  - A new gL3 page is added until **the last entry of a gL3 page is used**



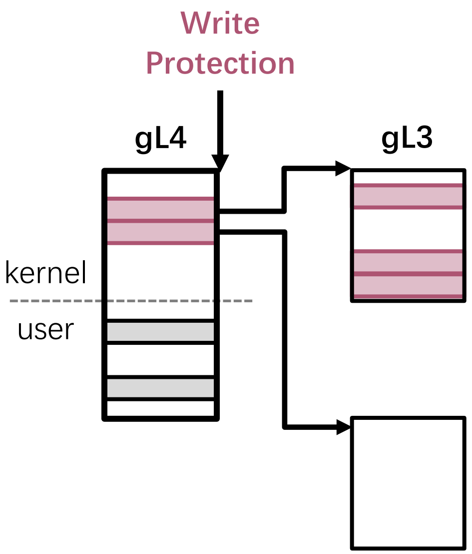
OPT-2. Trapping gL3 Instead of gL4

- Trap write access on kernel gL3 pages
  - A new gL3 page is added until **the last entry of a gL3 page is used**



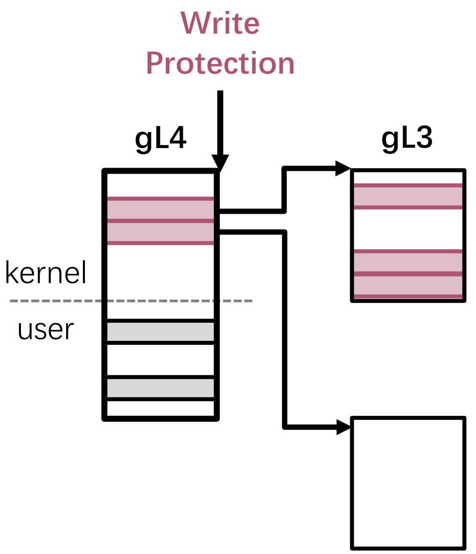
OPT-2. Trapping gL3 Instead of gL4

- Trap write access on kernel gL3 pages
  - A new gL3 page is added until **the last entry of a gL3 page is used**
- Trap write access on gL4 page
  - When one gL3 page’ s last entry is used



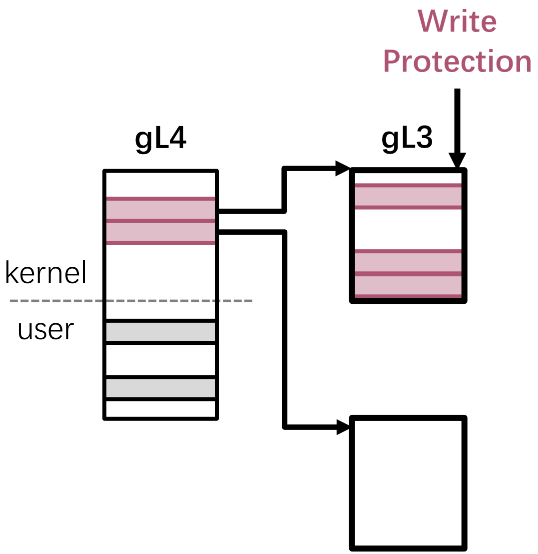
OPT-2. Trapping gL3 Instead of gL4

- Trap write access on kernel gL3 pages
  - A new gL3 page is added until **the last entry of a gL3 page is used**
- Trap write access on gL4 page
  - When one gL3 page' s last entry is used



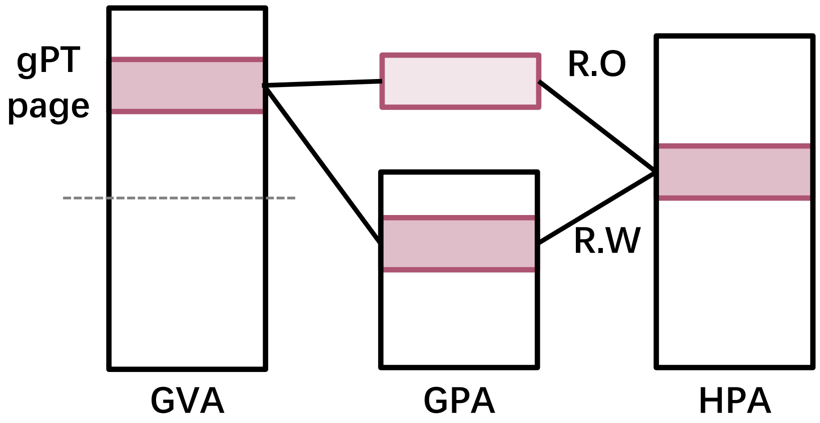
OPT-2. Trapping gL3 Instead of gL4

- Trap write access on kernel gL3 pages
  - A new gL3 page is added until **the last entry of a gL3 page is used**
- Trap write access on gL4 page
  - When one gL3 page's last entry is used
- Kernel rarely adds new gL3 page
  - One gL3 page maps 512GB memory region



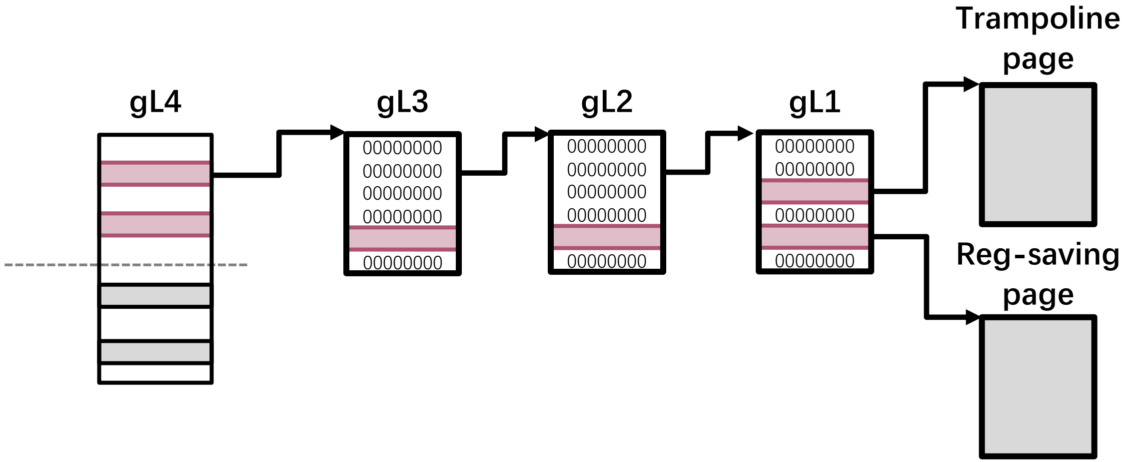
OPT-3. Setting gPT Access/Dirty-Bit

- Different access path between CPU and kernel
  - CPU accesses gPT by GPA
  - Kernel accesses gPT by GVA
- Construct different mapping for CPU and kernel access
  - Map gPT page's **GPA as R.W in EPT-k**
  - Map gPT page's GVA to **new GPA** and map the GPA as **R.O in EPT-k**



Trampoline

- Trampoline switches EPT at kernel enter/exit point
  - All kernel entries are stored in IDT or some specific MSRs
  - Exit point must contain specific instructions (e.g., sysretq)
- Map trampoline page in EPT-u
  - Two kernel pages in EPT-u
    - Trampoline code page
    - Reg-saving page



Seamless Protection

- Combing EPTI with live migration
  - I. Live migrate a VM to a host with EPTI
  - II. Construct EPT-k and EPT-u for the VM before resuming
  - III. Detect all kernel enter/exit points
  - IV. Inject trampoline with binary rewrite
  - V. Resume the VM

Malicious EPT Switching

- VMFUNC can be executed in user mode
  - Attacker can switch to EPT-K and perform Meltdown attack
- Make EPT-k useless in user mode
  - All memory except kernel code and kernel module are non-executable
  - **No instruction fetch** after switching to EPT-k in user mode

Evaluation

- Hardware platform
  - Intel Core i7-7700 (eight 3.6GHZ cores)
  - 16GB memory
- Software environment
  - Host Linux 4.9.75 + KVM
  - Guest Linux 4.9.75
- Guest environment
  - 4 vCPU (each vCPU is pinned on one physical core)
  - 8GB memory

VMFUNC vs. MOV to CR3

- Instruction cycle
  - VMFUNC: ~160 cycles
  - MOV to CR3: ~300 cycles
- TLB behavior
  - EPT switching **does not flush TLB**

Action	**Access again in EPT-0**	**Access again in EPT-1**
Invalid both EPTs' TLBs then fill EPT-0's TLB	3-5 cycles	120+ cycles
Fill both EPTs' TLBs then write CR3 in EPT-0	120+ cycles	120+ cycles
Fill both EPTs' TLBs then *invlpg* in EPT-0	120+ cycles	120+ cycles

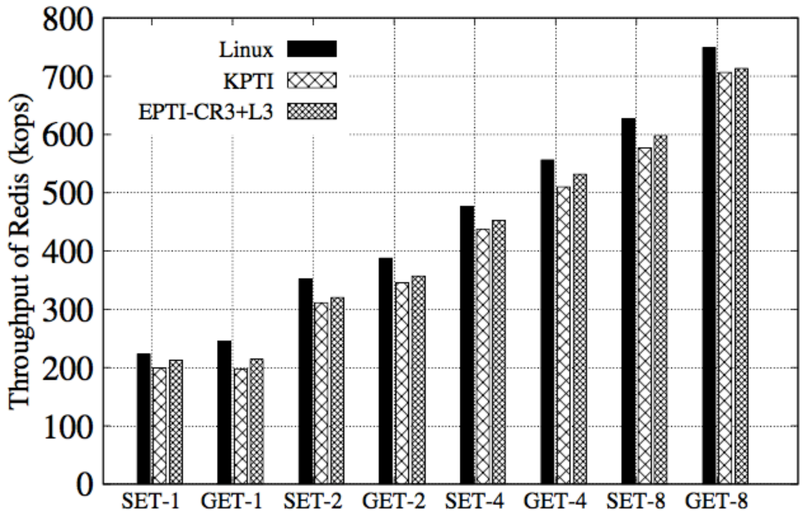
Micro-benchmark

- Lmbench

Operation (μs)	Linux	KPTI	EPTI-No	EPTI-CR3	EPTI-CR3+L3
Null syscall	0.04	0.16	0.12	0.12	0.12
Null I/O	0.07	0.2	0.17	0.17	0.16
Open/Close	0.70	0.93	0.84	0.84	0.83
Signal Handle	0.68	0.81	0.76	0.76	0.76
Fork syscall	72.9	79	80	80	75
Exec syscall	212	243	242	234	221
ctsw 16P/64K	6.07	7.37	7.66	7.66	6.39

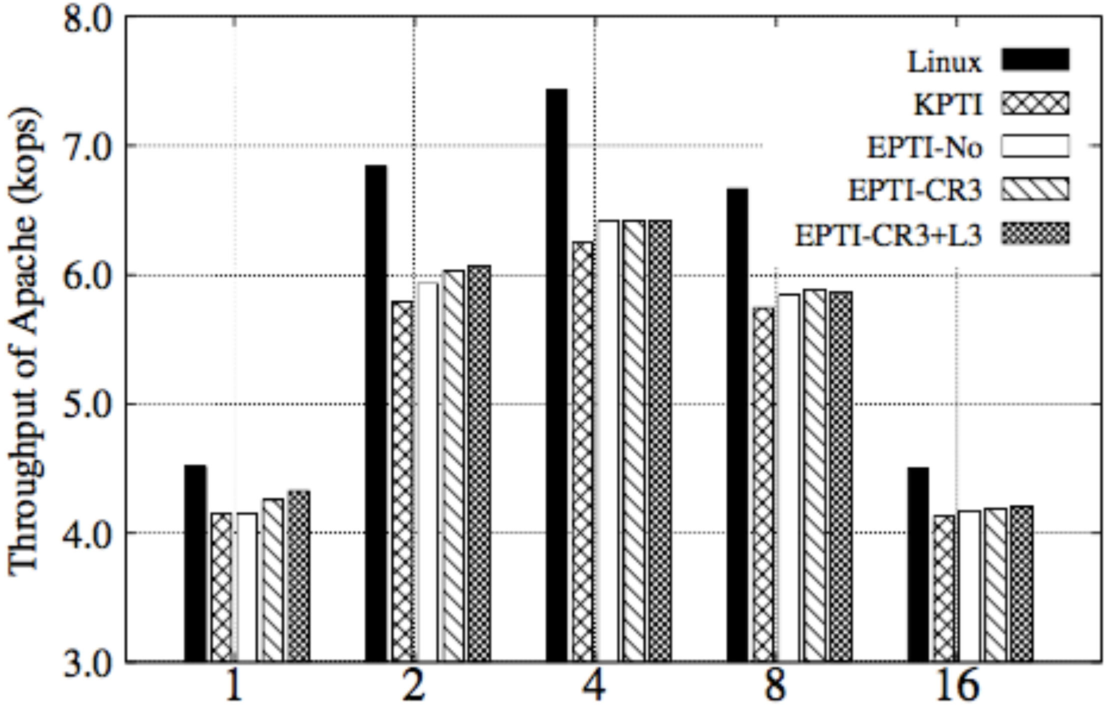
Application Overhead

- Redis throughput
  - Average overhead: KPTI 12%, EPTI 6%
  - Worst case: **KPTI 20%, EPTI 12%**



Application Overhead

- Apache throughput
  - KPTI 15%-18%
  - EPTI ~10%



EPTI Optimization

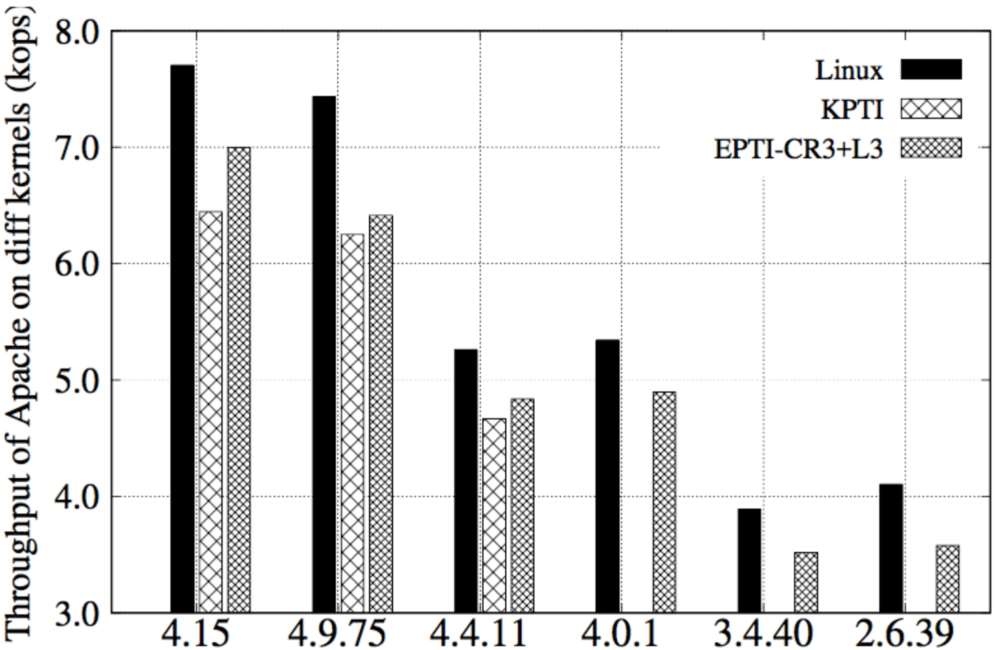
- Load CR3 works for frequently switching between limited CR3 values (e.g., apache)
- Trapping gL3 reduces all the VMExits

Benchmark	EPTI-No	EPTI-CR3	EPTI-CR3+L3
Redis 1-thread	540	464	0
Redis 8-thread	385	315	0
Apache 4-thread	45406	225	0
Apache 32-thread	40149	623	0
Compile Kernel -j8	609659	551023	0

EPTI-NO : A/D-bit
EPTI-CR3: A/D-bit + load CR3
EPTI-CR3+L3: all opts

Different Kernel Versions

- Apache throughput of different Linux versions
  - In Linux 4.15 (PCID enabled)
    - **KPTI 17%**
    - **EPTI 10%**



Conclusion

- Providing a new Meltdown defense method
- Protect **unmodified** guest VM
  - Work on different kernel versions
- **Seamless** protection
  - Without guest rebooting
- **Low** performance overhead

Thanks

Institute of Parallel And Distributed Systems (IPADS)
http://ipads.se.sjtu.edu.cn