Foreshadow-VMM: Breaking Virtual Machines Isolation

DEFCON ROME MEETUP – 22/02/2019

MARCO.SPAZIANI.BRUNELLA@UNIROMA2.IT

2018: The year of disaster

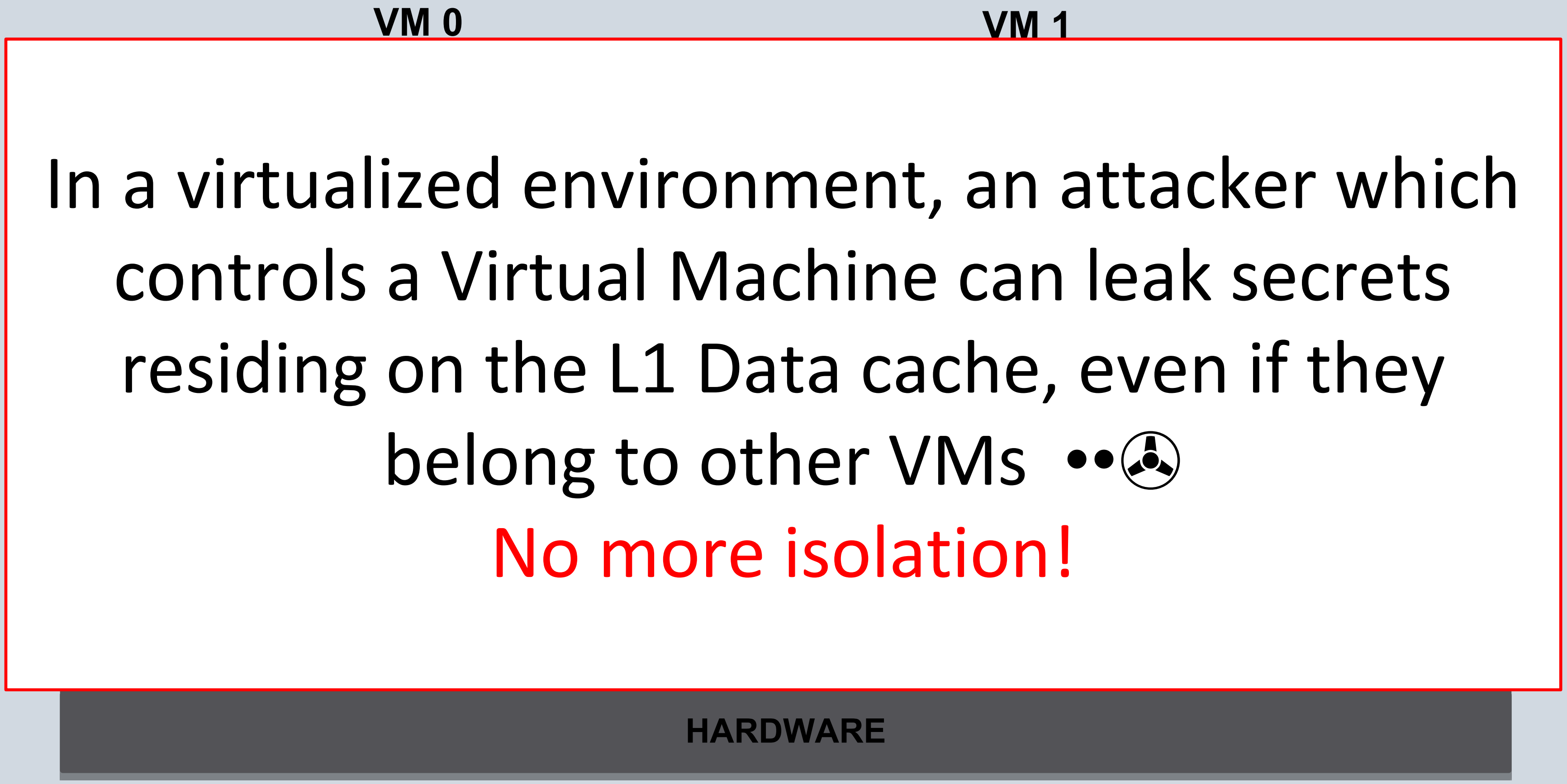
Spec
(CVE-

CVE-

K
R

Until Foreshadow-VMM, μarchitectural attacks were focused on (just) leaking data between processes running on the same machine.

Foreshadow-VMM:
The Big Picture



In a virtualized environment, an attacker which controls a Virtual Machine can leak secrets residing on the L1 Data cache, even if they belong to other VMs ••☢

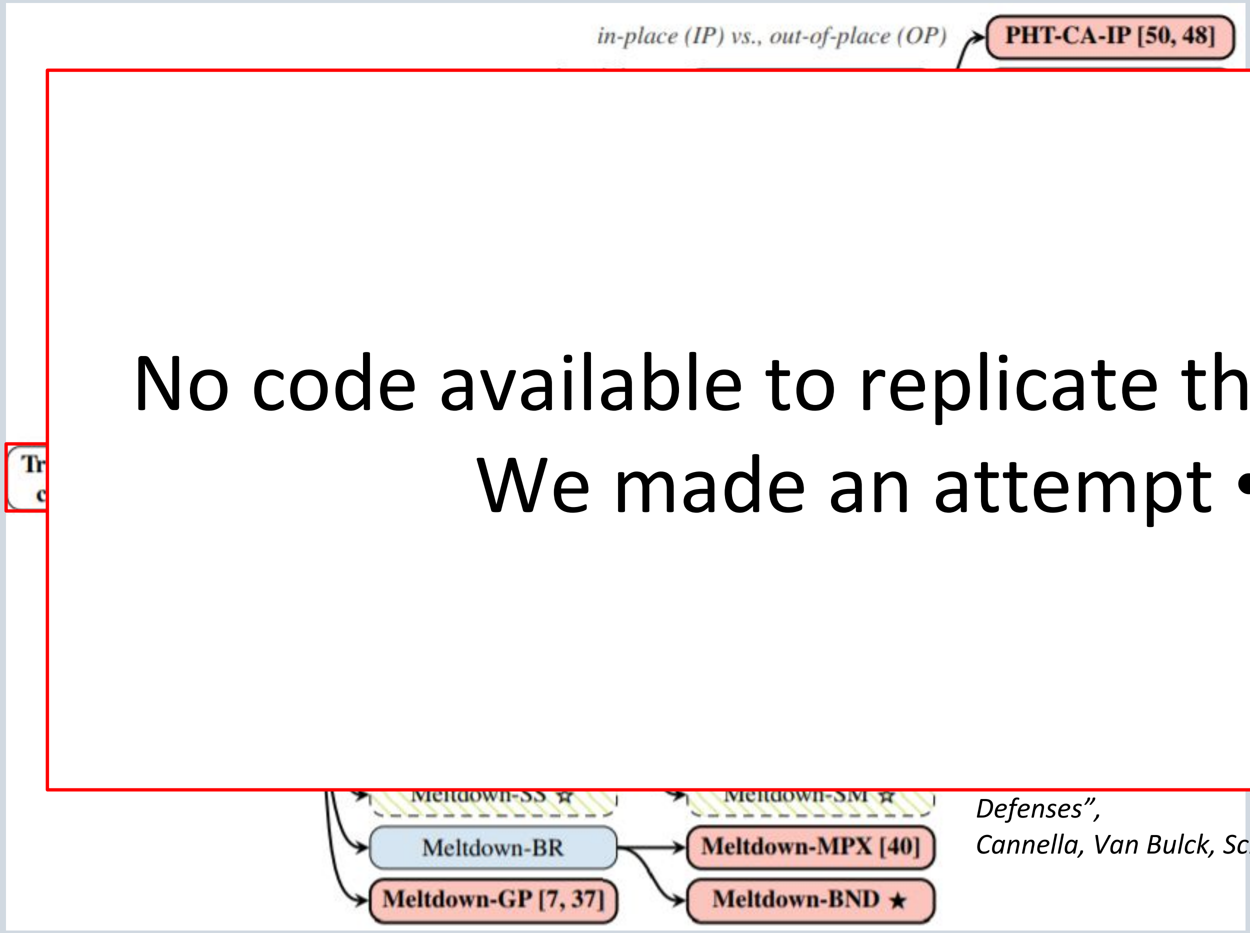
No more isolation!

Why is a problem?

- Virtualization has been an enabler for things such as cloud computing
- Moreover, is a foundation point of the emerging 5G network, in wich the network is «sliced» into virtual network functions
- Breaking the isolation in such environments means that data of different customers can be mixed up

•☎

Classification of the attack



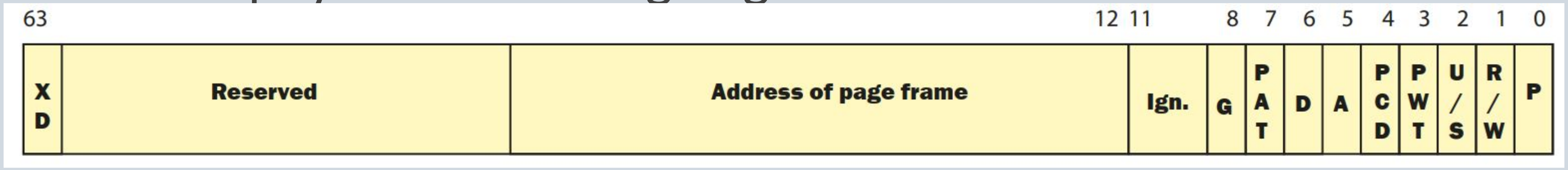
No code available to replicate the attack ••✇

We made an attempt •✆

Defenses”,
Cannella, Van Bulck, Schwarz, Lipp , et al…

Key Concepts 1/2

- The attack is focused on virtualized environments ••✇ multiple virtual machines running on the very same hardware (thus sharing CPUs, memory, etc..)
- Processes running on an Operating System do not access directly the physical memory, but they reference a “virtual” one •✇ the OS takes care of translating the virtual memory to the physical one using Page Table Entries

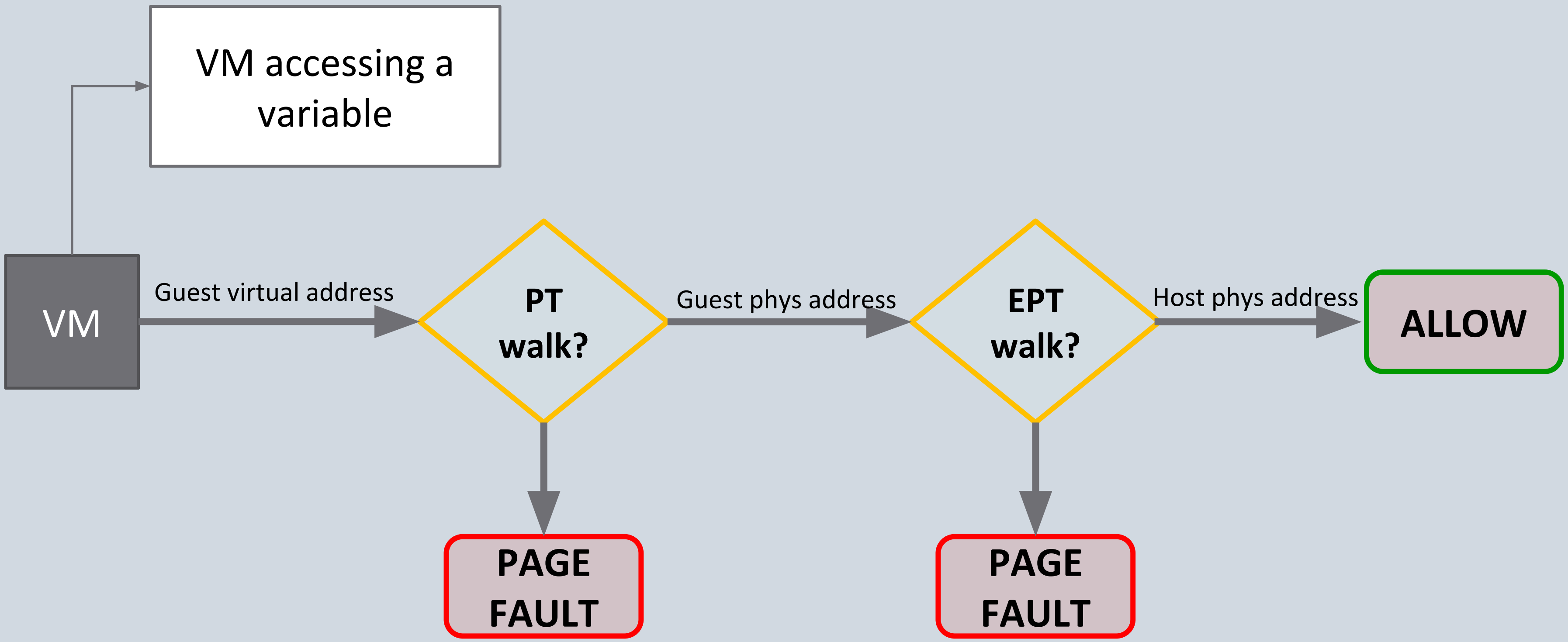


Intel x64 Page Table Entry

Key Concepts 2/2

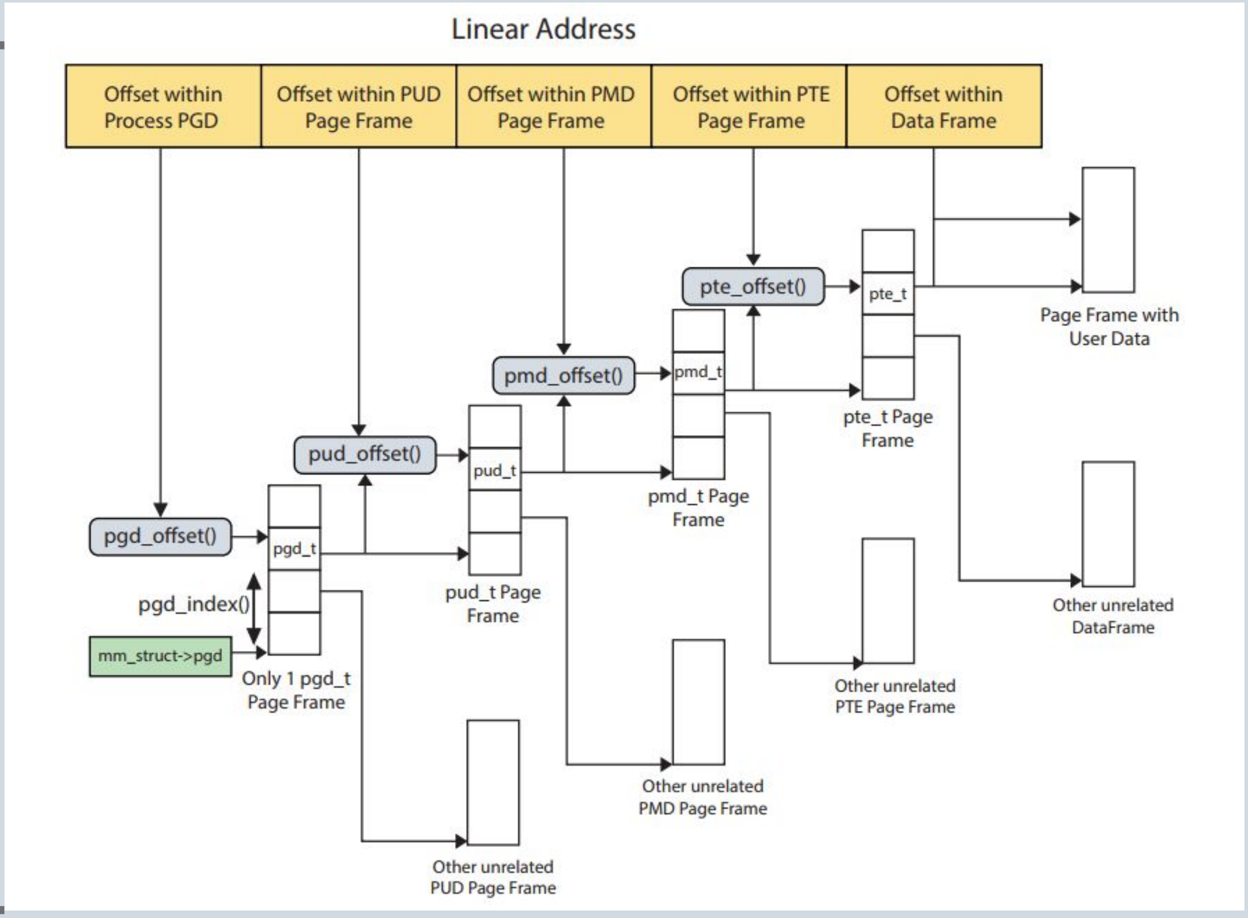
- Virtual machines that are running on the same CPU core share the L1-Data cache ••☢ Limitation of the attack!
- The attacker can use that as covert-channel to steal information from the victim’s address space

How address translation works

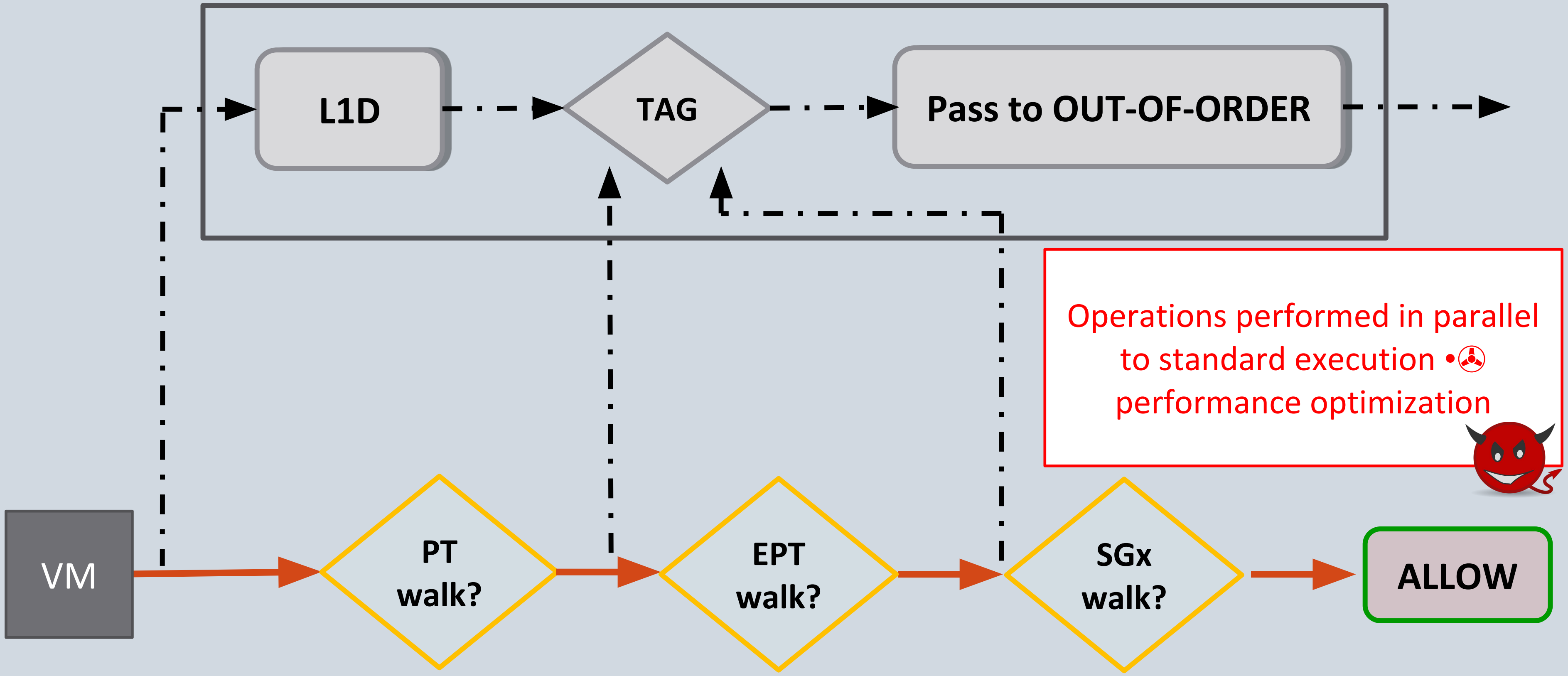


Page Table Walk in Linux

PT walk?


Linear Address
Offset within Process PGD
Offset within PUD Page Frame
Offset within PMD Page Frame
Offset within PTE Page Frame
Offset within Data Frame
pte_offset()
pte_t
Page Frame with User Data
pmd_offset()
pmd_t
pte_t Page Frame
pud_offset()
pud_t
pmd_t Page Frame
pgd_offset()
pgd_t
pud_t Page Frame
pgd_index()
mm_struct->pgd
Only 1 pgd_t Page Frame
Other unrelated DataFrame
Other unrelated PTE Page Frame
Other unrelated PMD Page Frame
Other unrelated PUD Page Frame

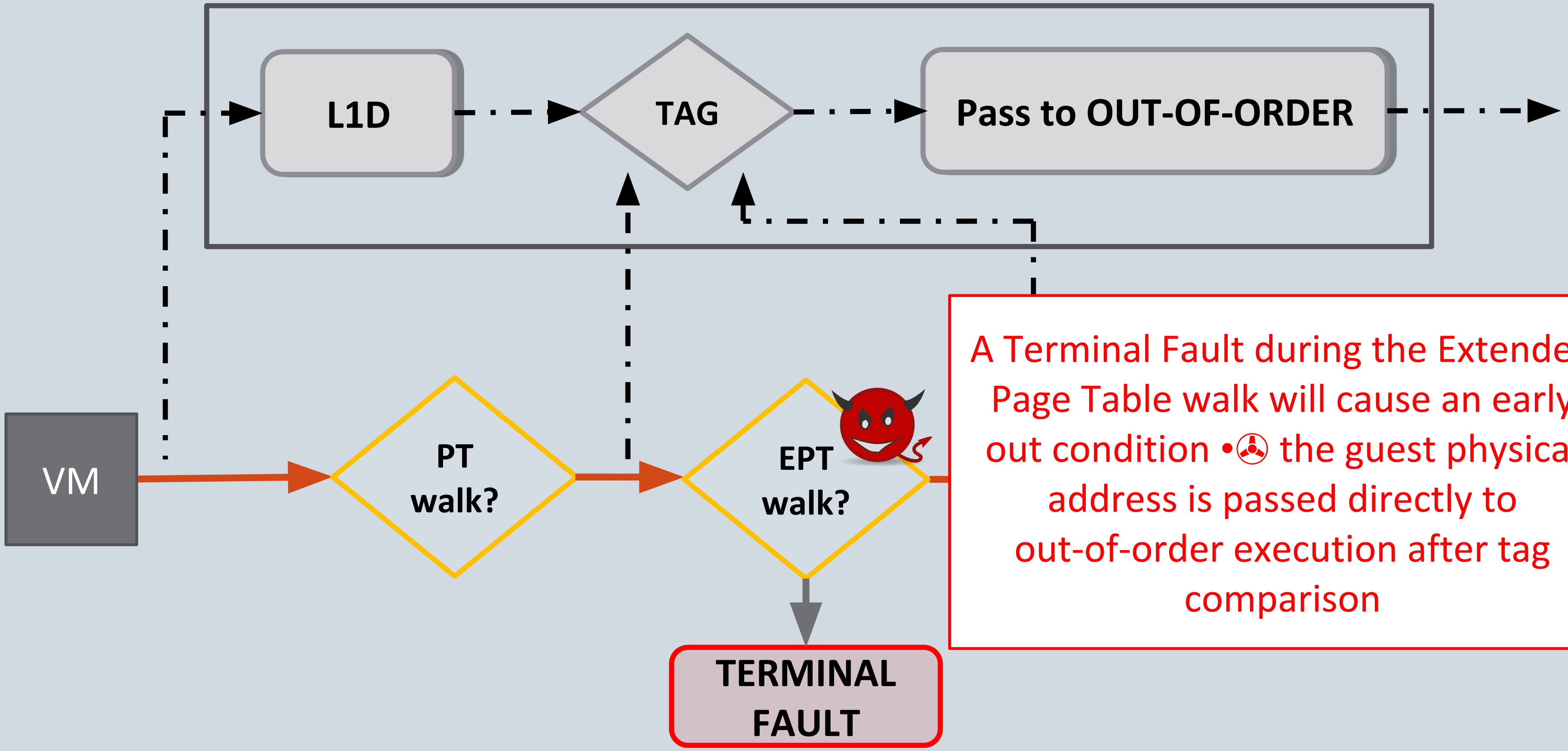
How Intel does that



Operations performed in parallel to standard execution •☢ performance optimization

"Intel SGX Explained", Costan, Devadas, MIT
"FORESHADOW: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution", Van Bulck et al

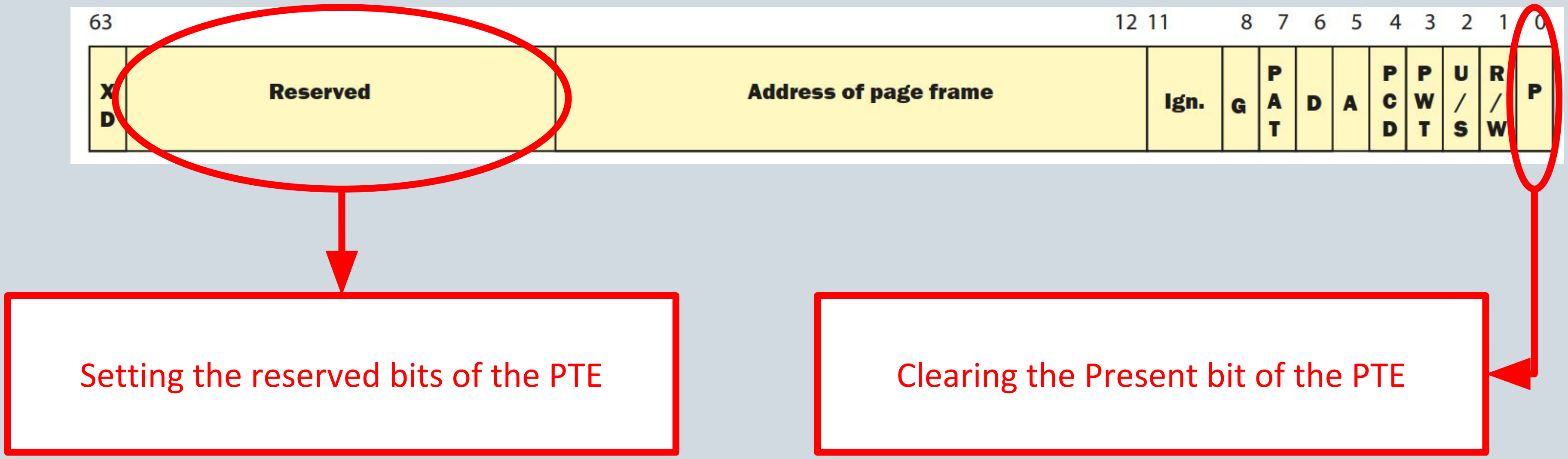
How Foreshadow-VMM works



A Terminal Fault during the Extended Page Table walk will cause an early out condition •☢ the guest physical address is passed directly to out-of-order execution after tag comparison

How to trigger a Terminal Fault

A terminal fault is a particular type of page fault that can be triggered by:



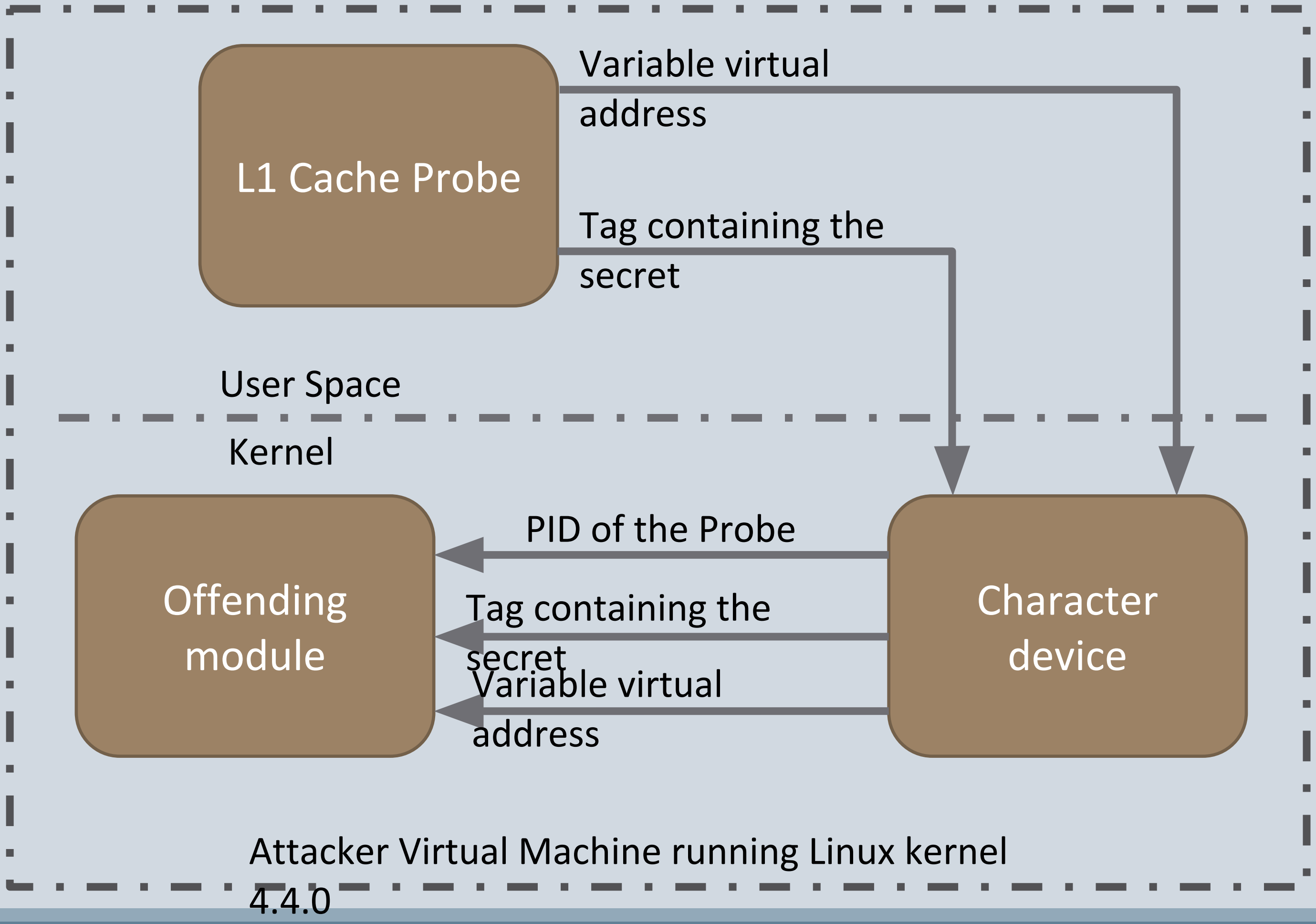
Setting the reserved bits of the PTE

Clearing the Present bit of the PTE

Steps to replicate the attack

1) Allocate a process wich will set up a commodity variable
2) Modify the page table entry of the variable by:
   a) Clearing the present bit of the PTE
   b) Substituting the tag with an arbitrary one, pointing to secret data
3) Access again the commodity variable, triggering a TF
4) Retrieve secret data by using Flush+Reload

Our approach



L1 cache Probe

- Is continuously flushing the cache, performing Flush+Reload attack to seek for an active cache line
- Is continuosly accessing the commodity variable, raising a terminal fault when the relevant PTE will be tampered by the offending module
- It communicates with the offending module trough a character device, passing the commodity variable virtual address and the new tag to install inside the PTE

Offending Module

- Is responsible for triggering the terminal fault (and more...)
- After receiving the PID of the probe, the vaddr of the commodity variable from the character device and the new tag to install, it performs the following steps:
  1) Page walk to reach the PTE of the commodity variable
  2) Clears the present bit of the commodity variable
  3) Substitutes the tag of the commodity variable
- When the probe will access again the commodity variable, it will raise a terminal fault

Linux and Terminal Faults

- When raising a Page Fault, Linux invokes the Page Fault Handler
- In the case of a TF, the Page Fault Hadler invokes the OOM-Killer, killing every process of the user (including the probe)
- We had to manage that in the offending mosule, intercepting the original Page Fault Handler and restoring the PTE before calling the latter •✇ the OS is not aware that the probe has encountered a TF

Live Demo

How has been mitigated?

- We need to identify the problem •☢ concurrent execution of two different machines on the same core •☢ Hyperthreading
- Many (simple) ways of attack mitigation, but huge price to pay in terms of performance loss:
  - Flush of the L1-D cache on every VMENTER
  - Disable Hyperthreading •⚐
- Need more efficient and transparent mitigations •☢ gang scheduling

Conclusions

- Very hard to replicate, even in this simple form!!!
- The attack was deduced on «paper», with no interaction with the OS
- Lot of noise on the cache due to the Page Fault Handler going nuts for the present bit cleared on the PTE
- Need to retrieve full strings and automate the attack
- Is this a problem in my datacenter? Of course yes ••