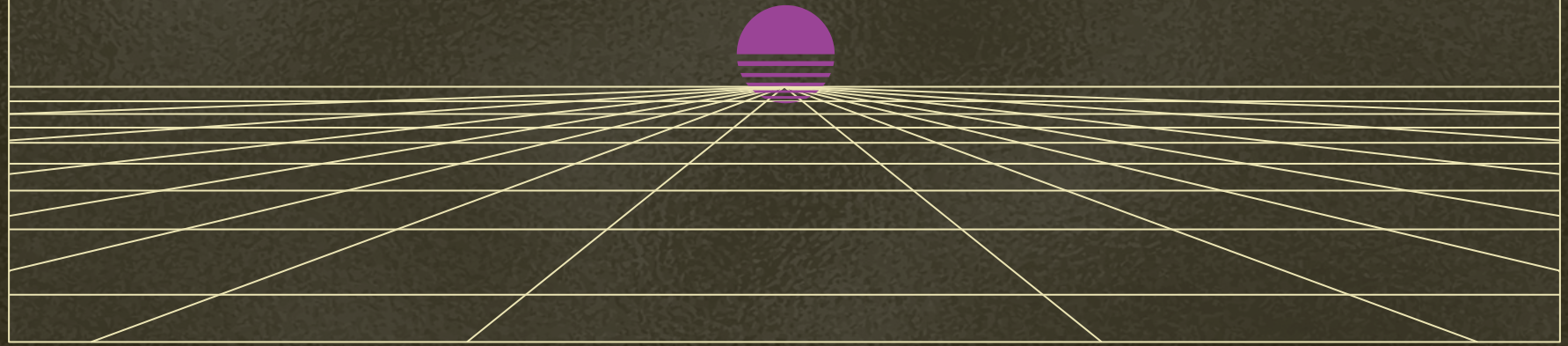
VIRTUAL MACHINE INTROSPECTION

ON ARM BASED SYSTEMS



Fernando C. Menardi

H2HC 2022

WHOAMI

“BORED ENGINEERING STUDENT”

Estag. Dev/Infra

➔ *Golang, Cloud, Docker and Kubernetes stuff*

Control Systems Engineer

➔ *(4.5/5)*

“Low level” stuff:

➔ Hardware, Reverse, Bin xpl..

➔ Tudo que faz beep e/ou pisca

github.com/Sargastico

@sargx90

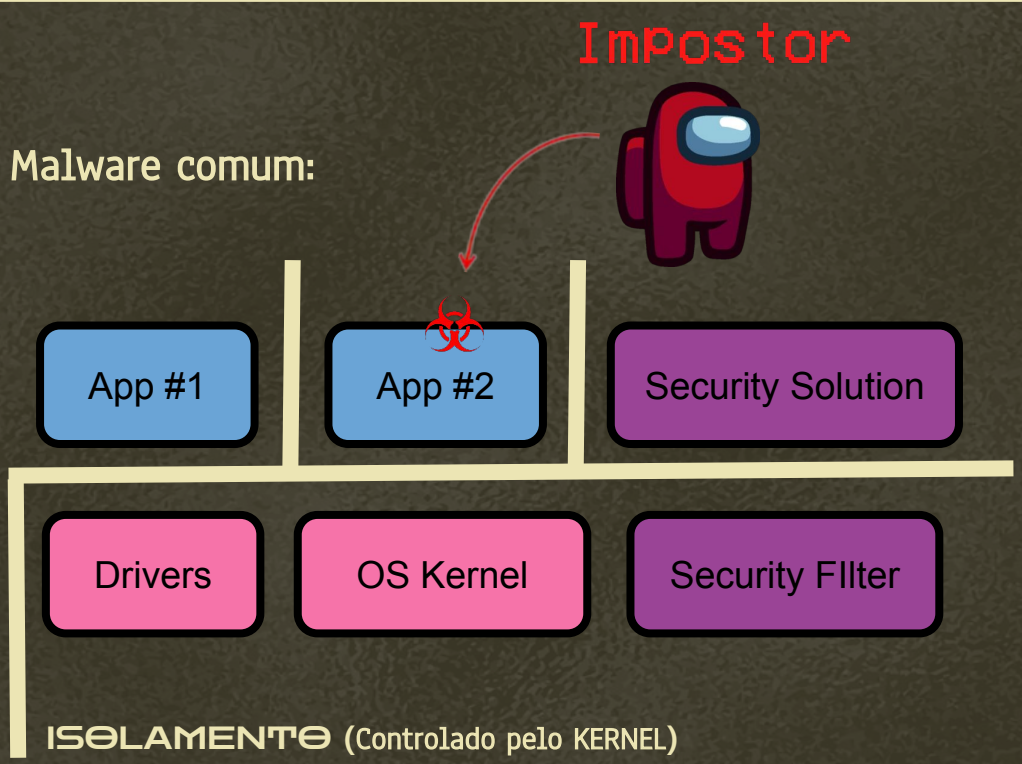
/in/fernando-menardi

sargx#0654

AGENDA

- **MOTIVAÇÃO**
  - PROBLEMA
  - SOLUÇÃO
- **HYPERVISORS**
  - Histórico em Infosec
  - Tipos
- **ARM**
  - RISC Arch
  - Exception Levels
  - Exception Vector Table
- **HYPERVISORS + ARM**
  - Virtual Memory
  - SLAT
- **XEN PROJECT**
  - Arquitetura
  - LibVMI
- **SYSCALL HOOKING**
  - Inline Hooking
  - SMC Trick
  - Event Channel
- **HIDING IN THE SHADOWS**
  - XEN Altp2m
  - Cache Incoherence
- **OQ FALTOU**
- **RECOMENDAÇÕES**
- **REFERÊNCIAS**

PR0BL3M4


Impostor
Malware comum:
App #1
App #2
Security Solution
Drivers
OS Kernel
Security FIlter
ISOLAMENTO (Controlado pelo KERNEL)

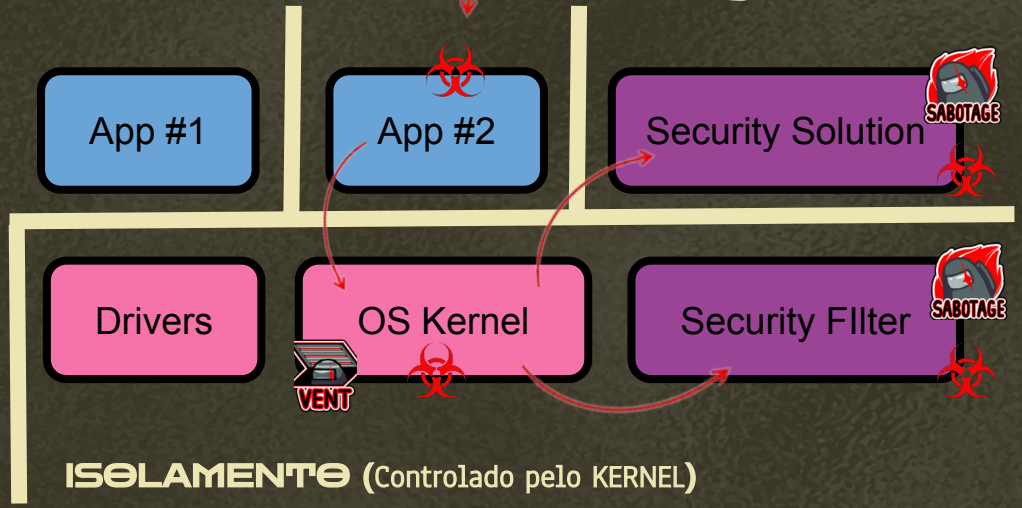
PR0BL3M4

Impostor

Malware Avançado:
(aka rootkits e seus amigos)

Malware é executado no mesmo contexto e com os mesmos privilégios dos softwares anti-malware (AVs, EDRs e etc)

"OBSERVER EFFECT"



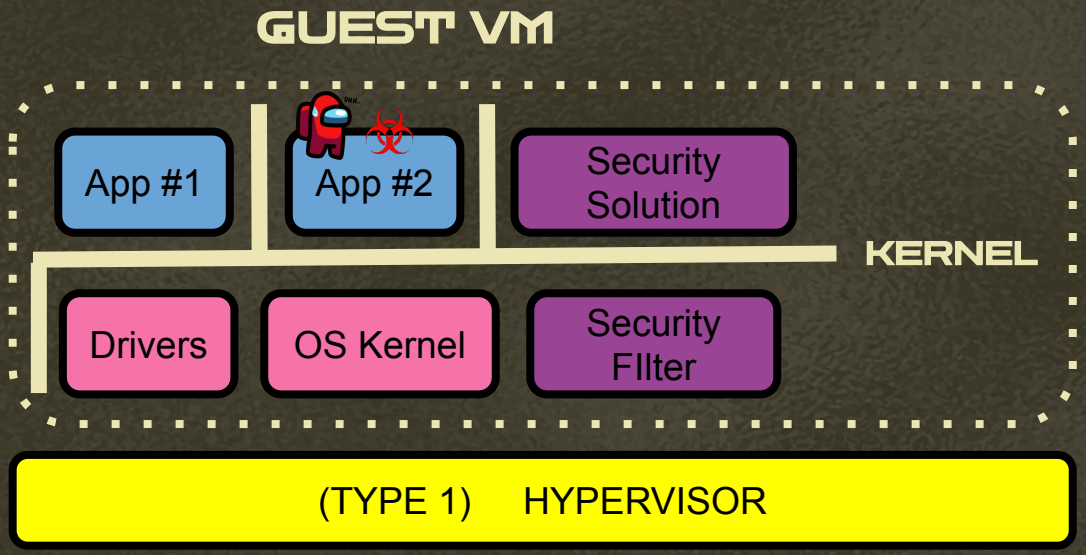
ISOLAMENTO (Controlado pelo KERNEL)


HACKERS CAN TURN YOUR HOME PAGER INTO A BOMB

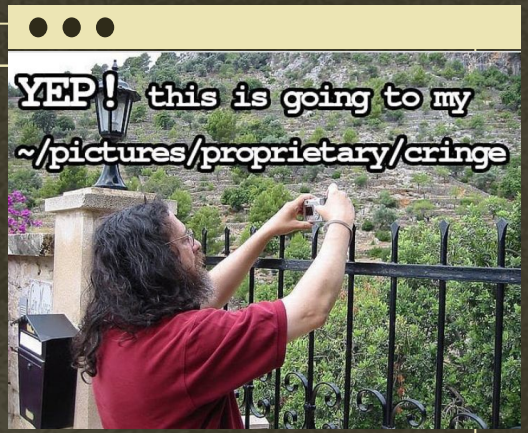
... & blow your family to smithereens!

Sickos can wreak death and destruction from thousands of miles away!

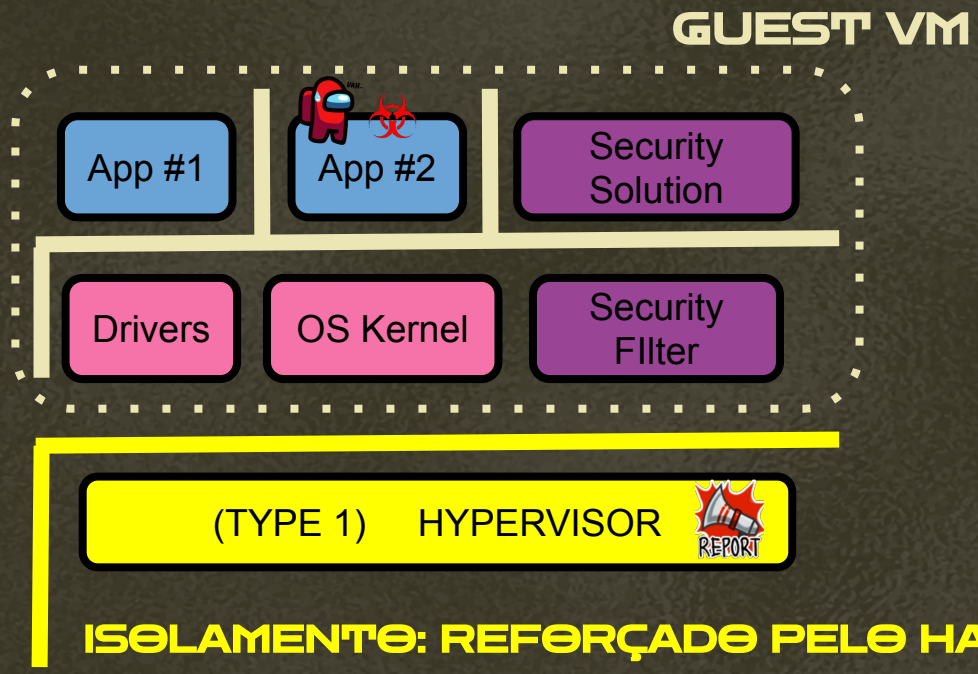
S0LUC40

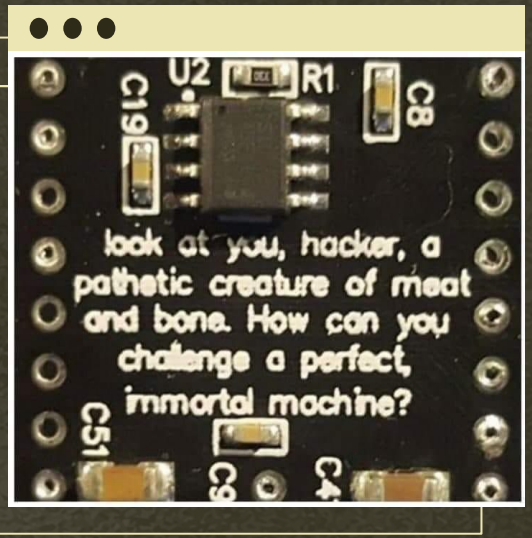


+1 LAYER (???)

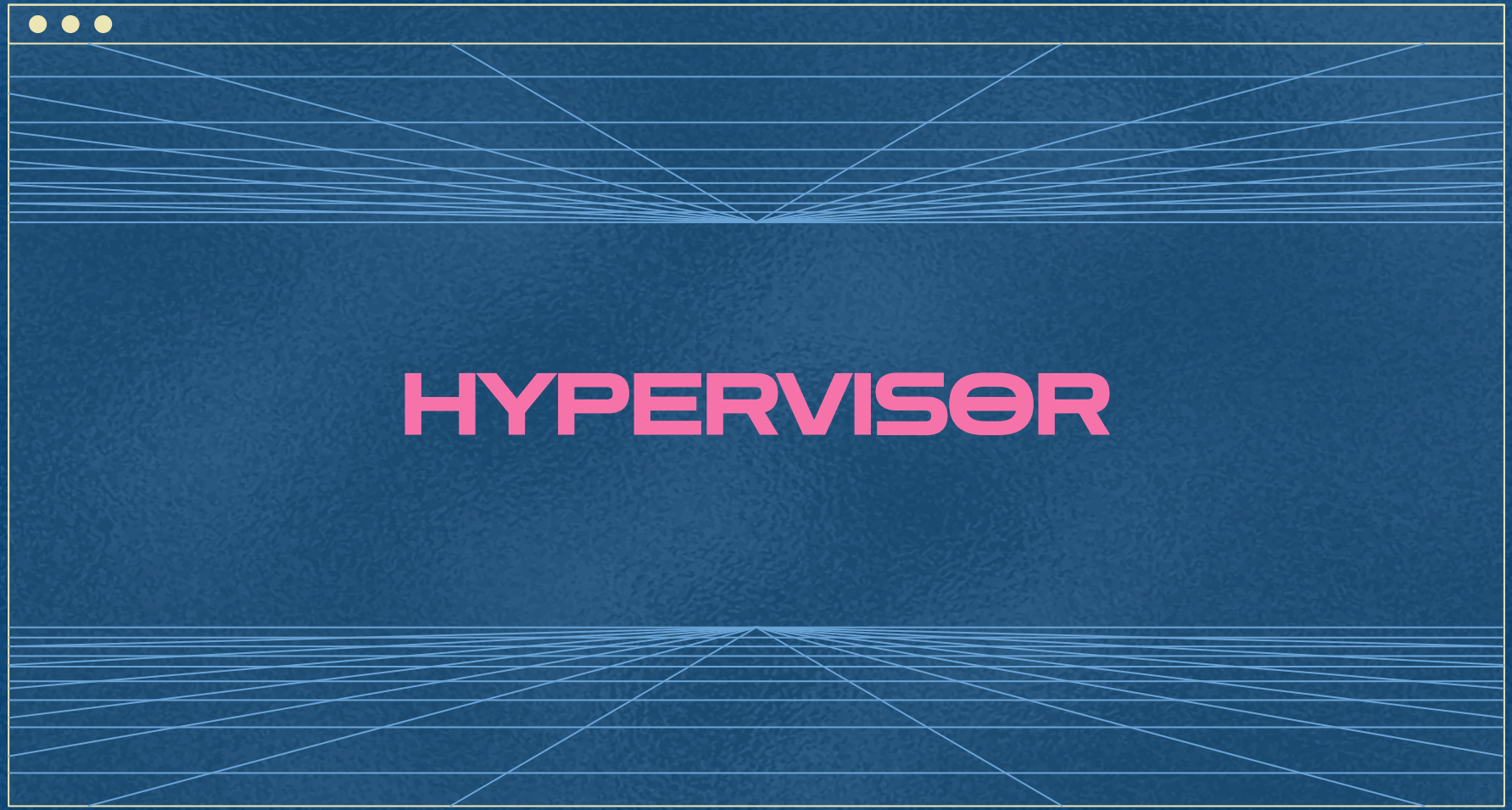


SOLUÇÃO

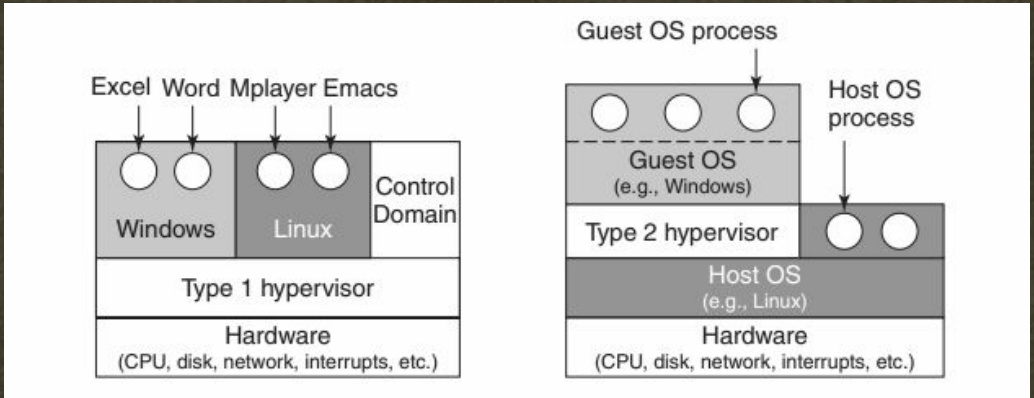




ISOLAMENTO: REFORÇADO PELO HARDWARE!


HYPERVISOR

TIPOS DE HYPERVISOR



TANENBAUM, Andrew S. Modern operating systems. Boston: Pearson, 2015.

Utilizando um Hypervisor Type-1, é possível reduzir uma camada de abstração.

SECURITY + HYPERVISOR

(x86 LAND)

issue : #69 | Release date : 2016-05-06 | Editor : The Phrack Staff

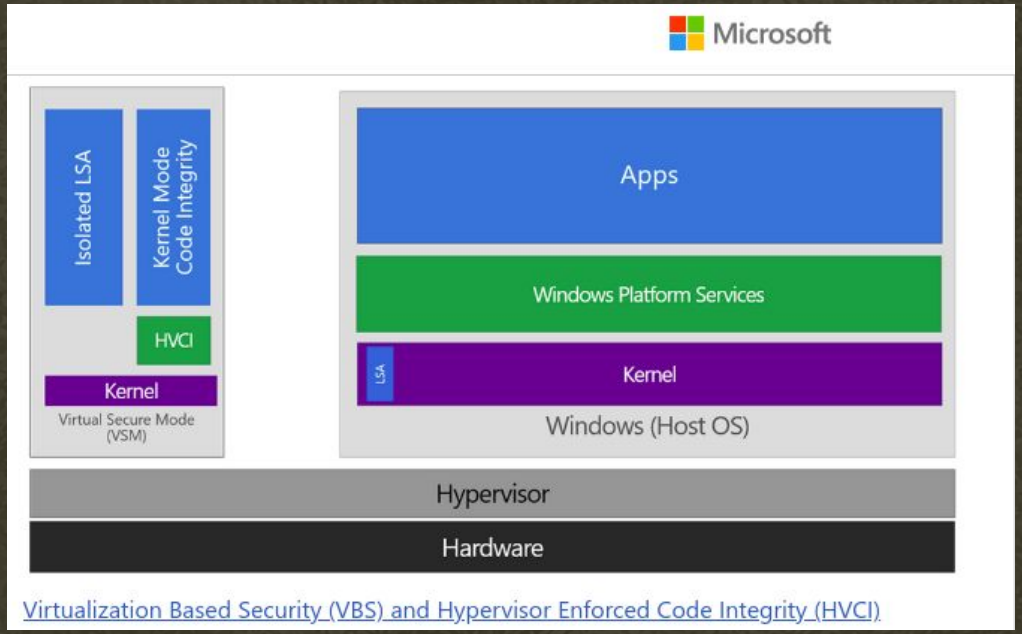
==Phrack Inc.==

Volume 0x0f, Issue 0x45, Phile #0x0f of 0x10

|=---=|
|=----------------------=[How to hide a hook]=-------------------------=|
|=--------------------=[A hypervisor for rootkits]=----------------------=|
|=---=|
|=------------------=[uty <whensungoes@gmail.com>]=---------------------=|
|=----------------=[saman <saman.zonouz@rutgers.edu>]=------------------=|
|=---=|

http://phrack.org/issues/69/15.html

SECURITY + HYPERVISOR



https://techcommunity.microsoft.com/t5/windows-insider-program/virtualization-based-security-vbs-and-hypervisor-enforced-code/m-p/240571

ARM

THE ADVANCED RISC MACHINE

> START?

ADVANCED RISC MACHINE

STOP DOING X86

- UARCHS WERE NOT SUPPOSED TO BE MICROCODED

- YEARS OF OPTIMIZATION yet NO REAL-WORLD USE FOUND for using anything other than LOAD, STORE and ADD

- wanted to use more complicated instructions for a laugh? We have a tool for that: it is called "COPROCESSOR 0"

- "Yes please MOV this to a location in memory. Please PUSH this to and POP that from the hardware stack" - Statements dreamed up by the utterly deranged.

LOOK at what the X86 companies have been demanding your respect for all this time, with all the manufacturing processes we designed for them

(These are REAL products, that were REALLY marketed)

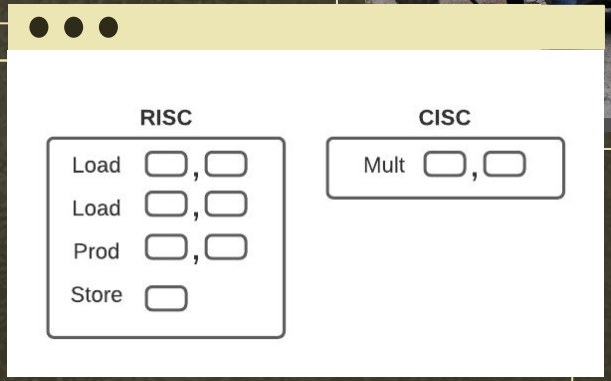
??????

????????

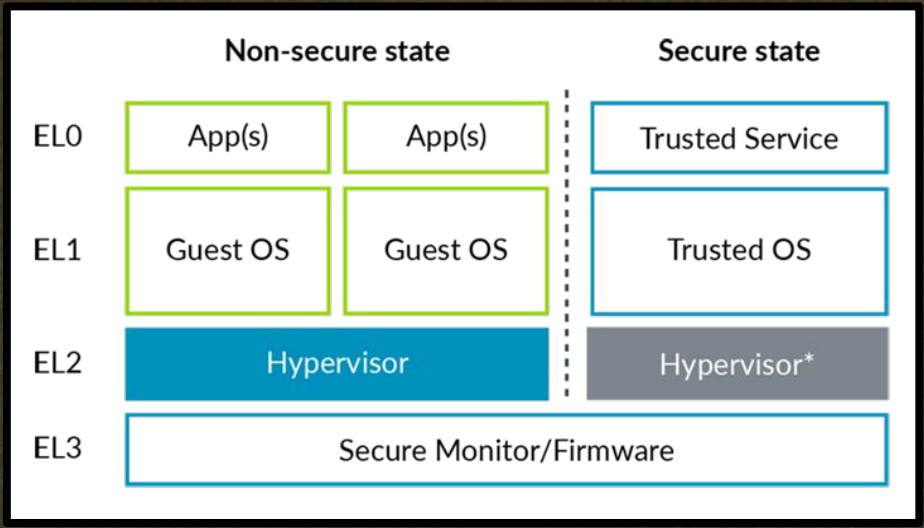
??????????????????

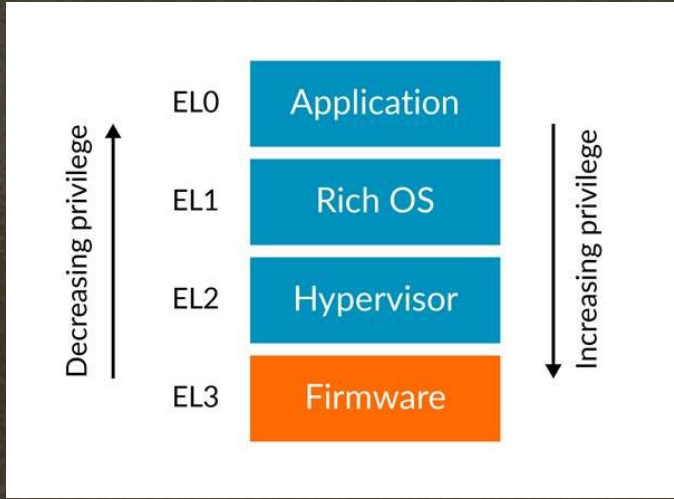
"Hello I would like you to PCLMULQDQ please"

They have played us for absolute fools



EXCEPTION LEVELS





https://developer.arm.com/documentation/102142/0100/Virtualization-in-AArch64

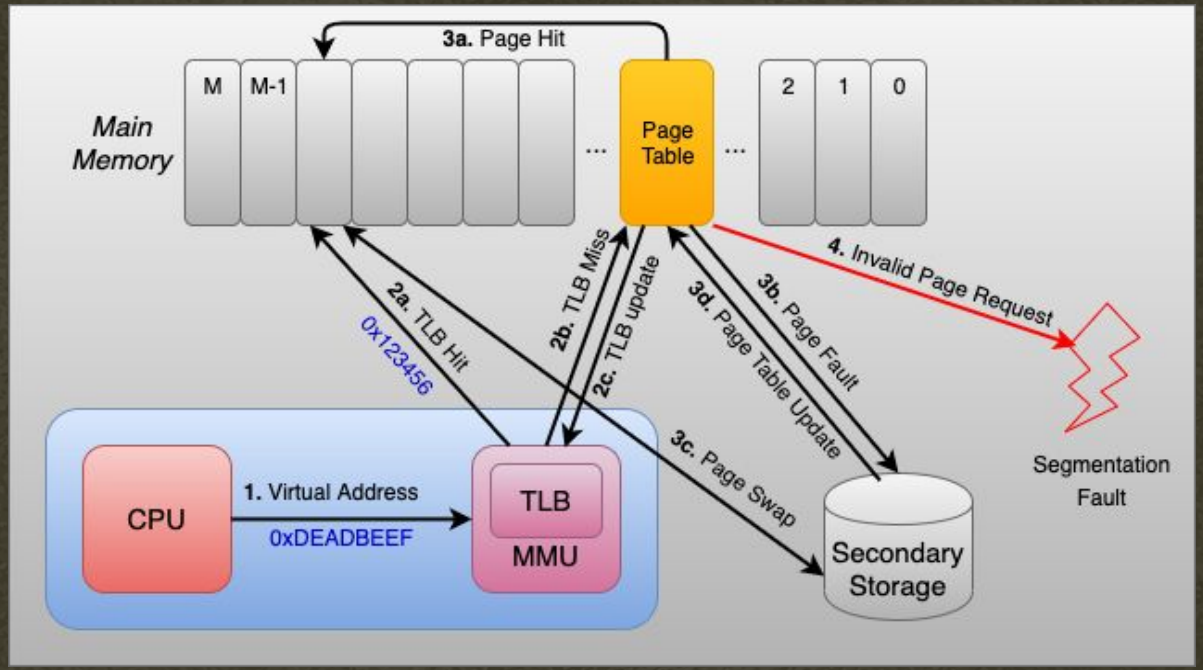
EXCEPTION VECTOR TABLE

	0x780	SError / vSError	Exception from a lower EL and all lower ELs are AArch32.
	0x700	FIQ / vFIQ	
	0x680	IRQ / vIRQ	
	0x600	Synchronous	
	0x580	SError / vSError	Exception from a lower EL and at least one lower EL is AArch64.
	0x500	FIQ / vFIQ	
	0x480	IRQ / vIRQ	
	0x400	Synchronous	
	0x380	SError / vSError	Exception from the current EL while using SP_ELx
	0x300	FIQ / vFIQ	
	0x280	IRQ / vIRQ	
	0x200	Synchronous	
	0x180	SError / vSError	Exception from the current EL while using SP_EL0
	0x100	FIQ / vFIQ	
	0x080	IRQ / vIRQ	
VBAR_ELn	+ 0x000	Synchronous	

https://developer.arm.com/documentation/100933/0100/AArch64-exception-vector-table

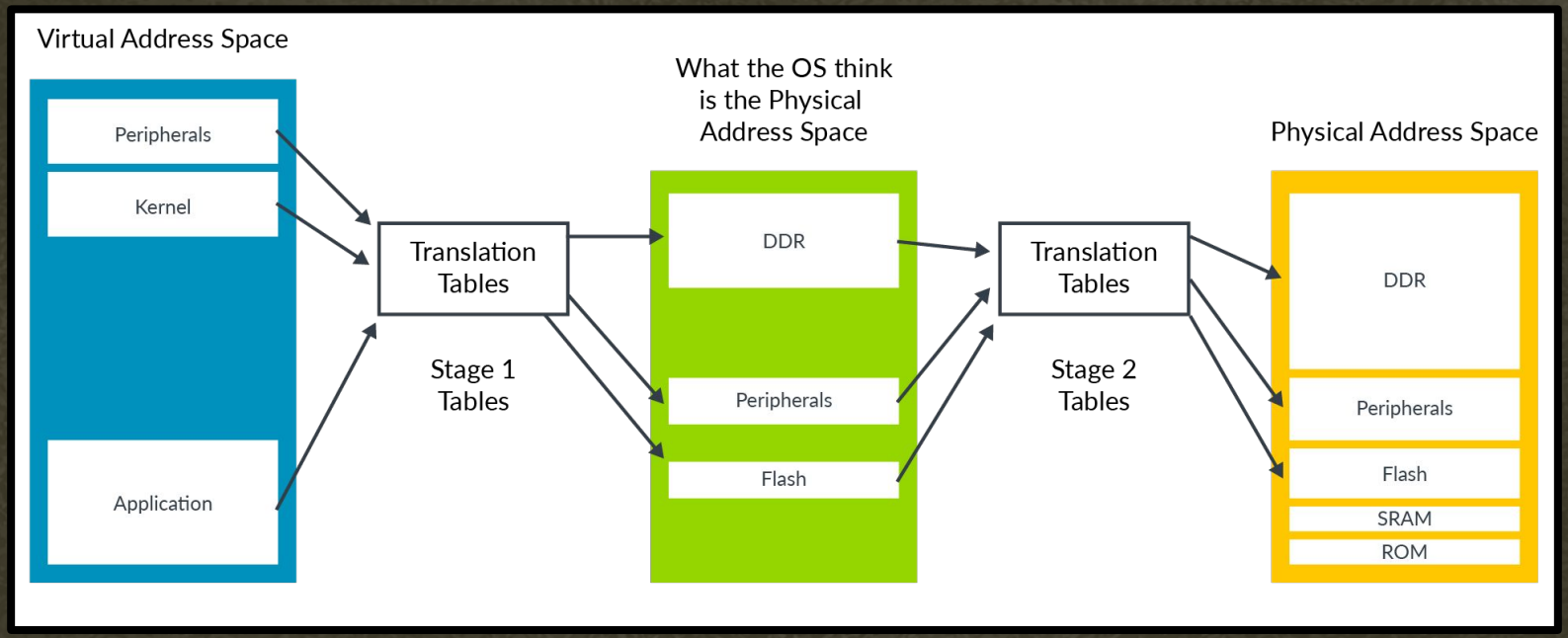
HYPERVISOR + ARM

VIRTUAL MEMORY



https://www.starlab.io/blog/deep-dive-mmu-virtualization-with-xen-on-arm

SLAT (SECOND LEVEL ADDRESS TRANSLATION)



https://developer.arm.com/documentation/102142/0100/Stage-2-translation

SLAT (SECOND LEVEL ADDRESS TRANLATION)

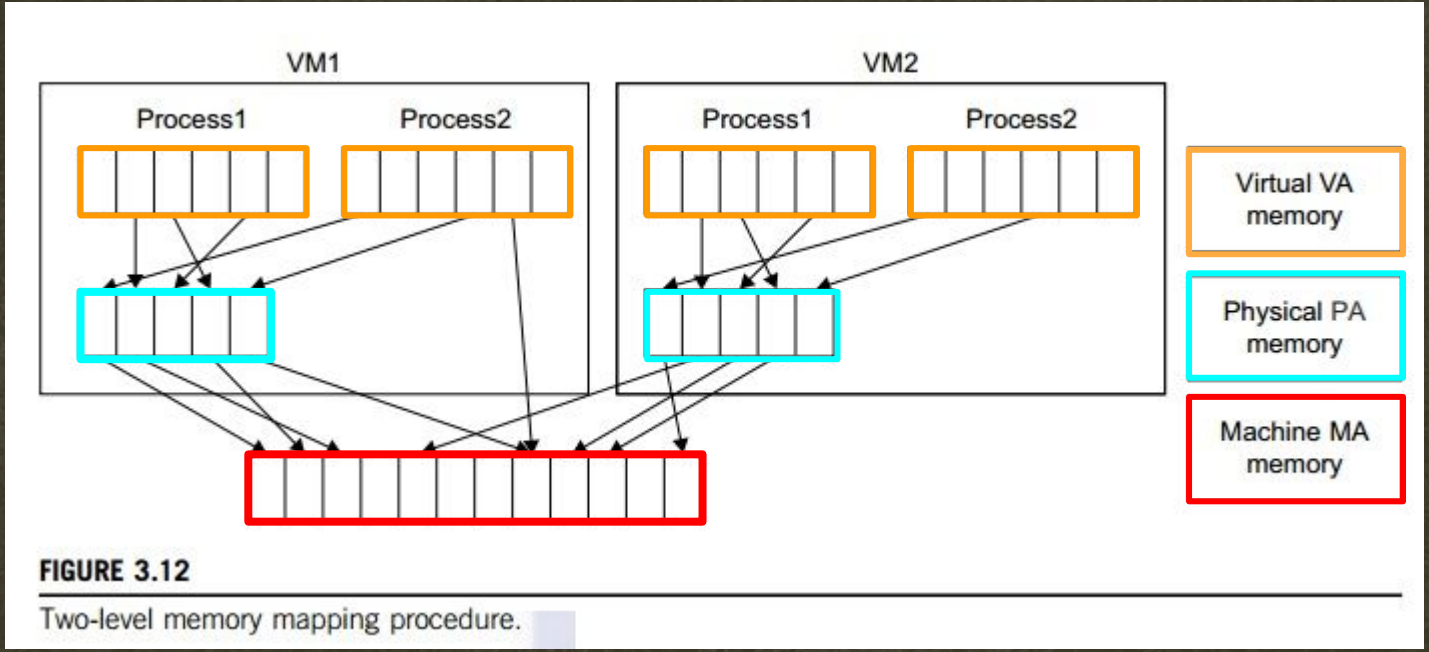


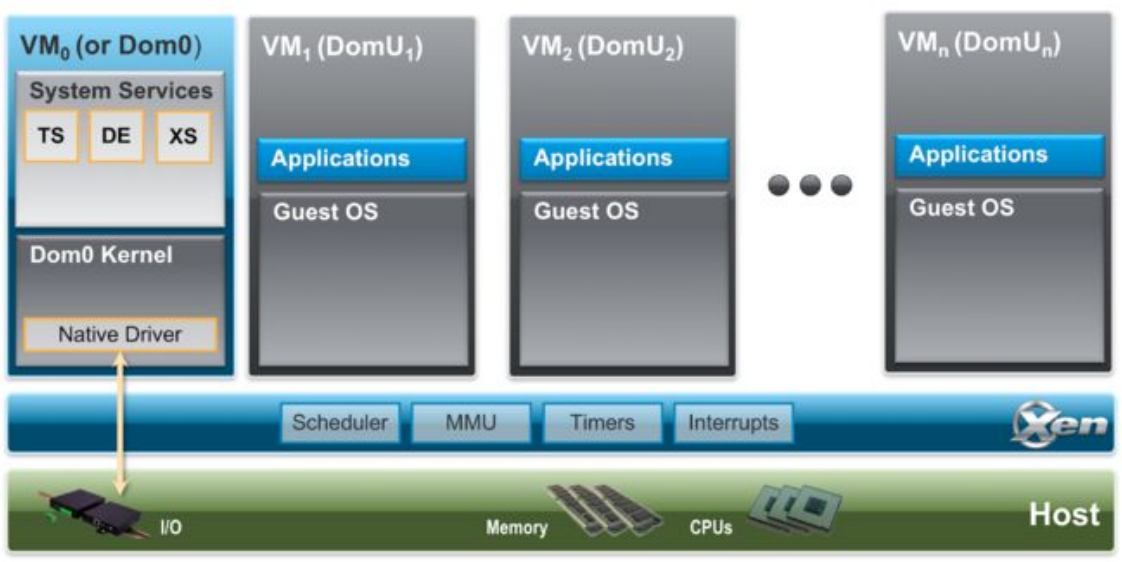
FIGURE 3.12

Two-level memory mapping procedure.

Adaptado de: https://www.brainkart.com/article/Memory-Virtualization_11340/

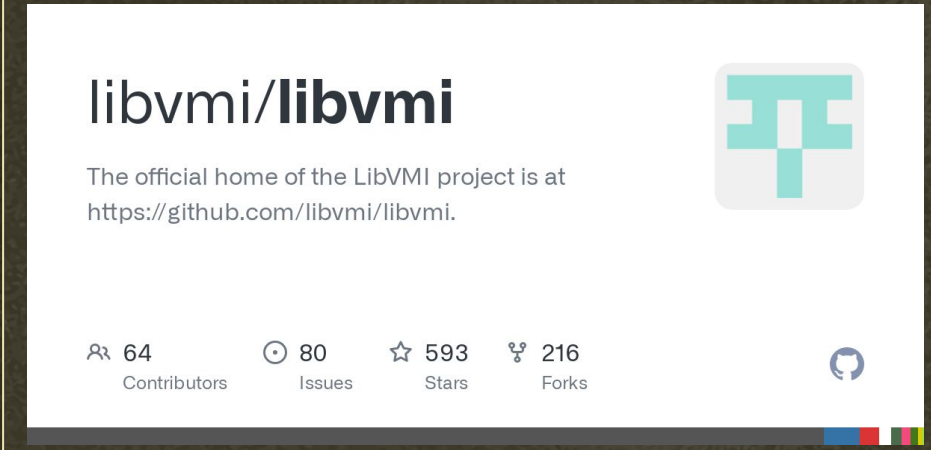
XEN HYPERVISOR

ARQUITETURA



https://wiki.xenproject.org/wiki/Xen_Project_Software_Overview

INSTRUMENTALIZAÇÃO



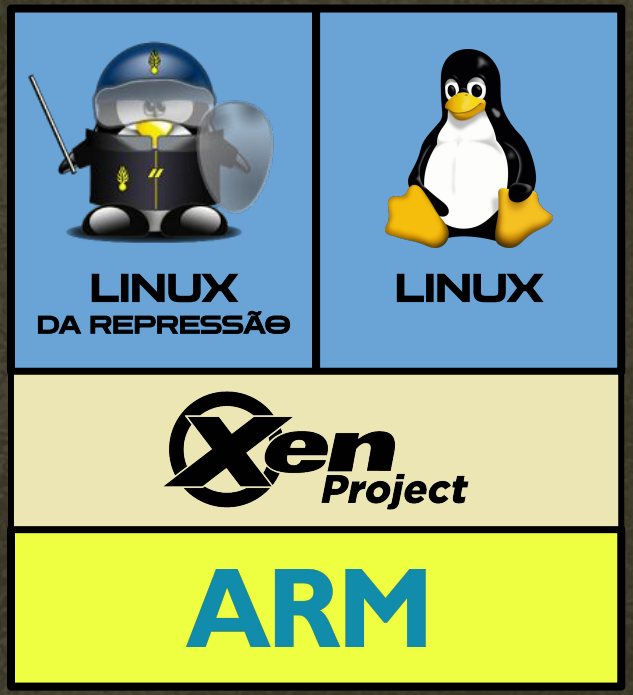
https://github.com/libvmi/libvmi

+

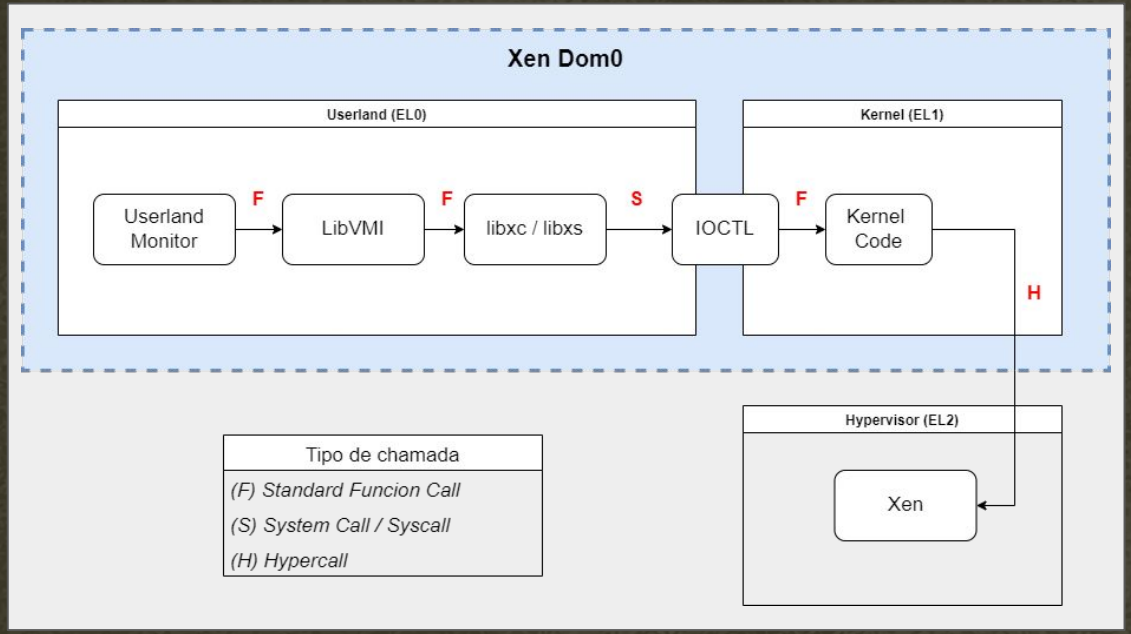
https://xenproject.org/

SOLUÇÃO


OH SHIT I TOOK
BOTH PILLS AND
NOW I'M STUCK IN
HYPERVISORS
SOLUTIONS


LINUX
DA REPRESSÃO
LINUX
Xen Project
ARM

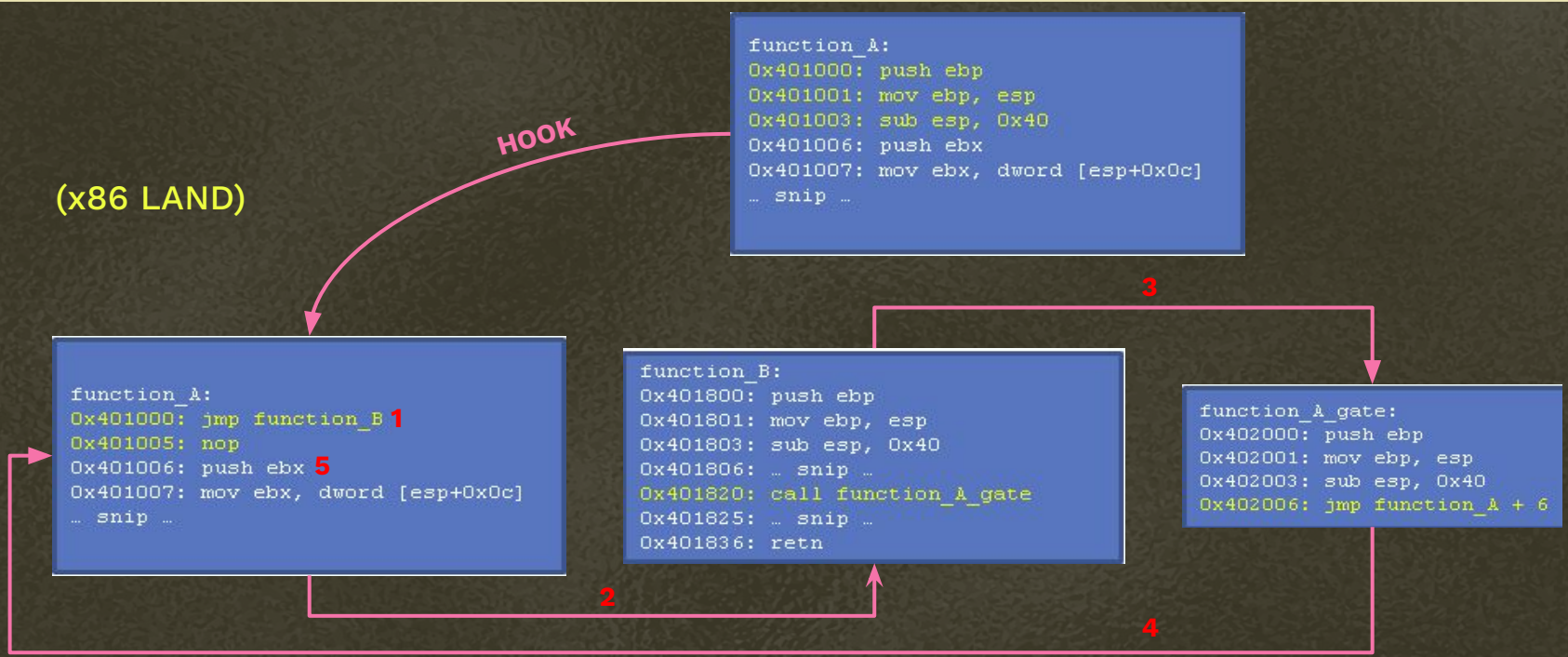
INSTRUMENTALIZAÇÃO



Elaboração própria.

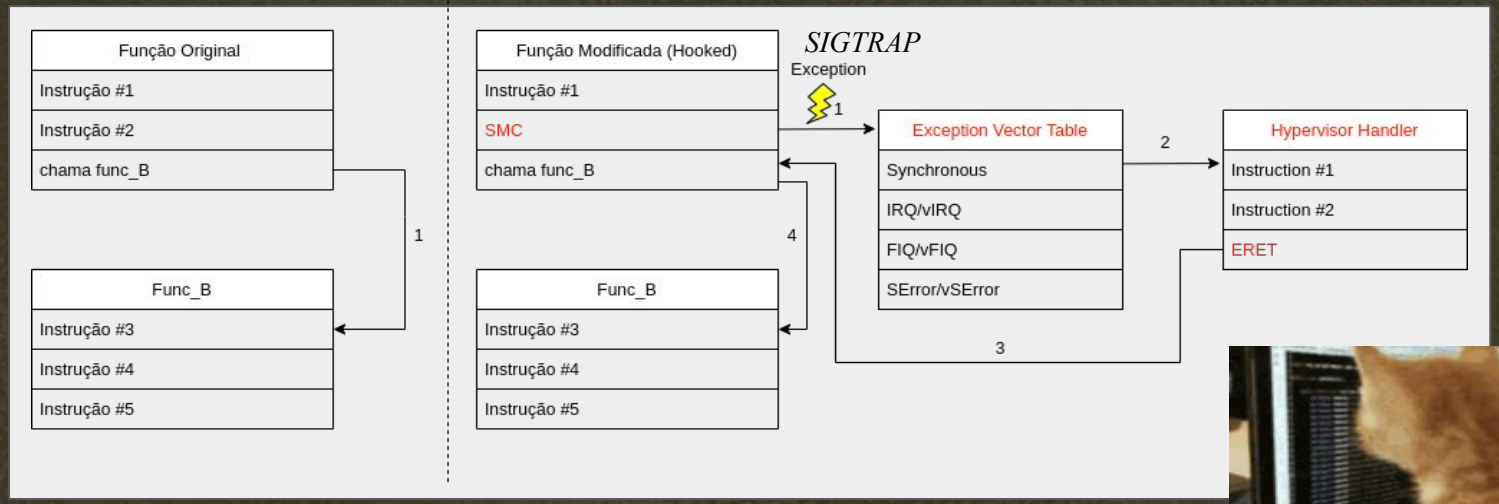
SYSCALL HOOKING

INLINE HOOKING



Adaptado de: http://jbremer.org/x86-api-hooking-demystified/#ah-abstract

SYSCALL HOOKING



Elaboração própria.

SMC = SECURE MONITOR CALL

só podem ser direcionadas para uma TEE no *TruztZone* ou para o *Hypervisor*

```
ENTRY(hyp_traps_vector)
        ventry  hyp_sync_invalid            /* Synchronous EL2t */
        ventry  hyp_irq_invalid             /* IRQ EL2t */
        ventry  hyp_fiq_invalid             /* FIQ EL2t */
        ventry  hyp_error_invalid           /* Error EL2t */

        ventry  hyp_sync                    /* Synchronous EL2h */
        ventry  hyp_irq                     /* IRQ EL2h */
        ventry  hyp_fiq_invalid             /* FIQ EL2h */
        ventry  hyp_error                   /* Error EL2h */

        ventry  guest_sync                  /* Synchronous 64-bit EL0/EL1 */
        ventry  guest_irq                   /* IRQ 64-bit EL0/EL1 */
        ventry  guest_fiq_invalid           /* FIQ 64-bit EL0/EL1 */
        ventry  guest_error                 /* Error 64-bit EL0/EL1 */

        ventry  guest_sync_compat           /* Synchronous 32-bit EL0/EL1 */
        ventry  guest_irq_compat            /* IRQ 32-bit EL0/EL1 */
        ventry  guest_fiq_invalid_compat    /* FIQ 32-bit EL0/EL1 */
        ventry  guest_error_compat          /* Error 32-bit EL0/EL1 */
```

```
void init_traps(void)
{
    /*
     * Setup Hyp vector base. Note they might get updated with the
     * branch predictor hardening.
     */
    WRITE_SYSREG((vaddr_t)hyp_traps_vector, VBAR_EL2);

    /* Trap Debug and Performance Monitor accesses */
    WRITE_SYSREG(HDCR_TDRA|HDCR_TDOSA|HDCR_TDA|HDCR_TPM|HDCR_TPMCR,
                 MDCR_EL2);

    /* Trap CP15 c15 used for implementation defined registers */
    WRITE_SYSREG(HSTR_T(15), HSTR_EL2);
```

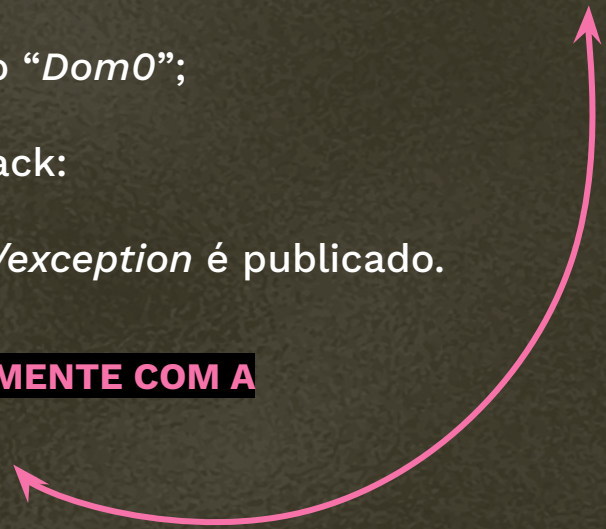
```
/* C entry point for boot CPU */
void __init start_xen(unsigned long boot_phys_offset,
                      unsigned long fdt_paddr)
{
    size_t fdt_size;
    int cpus, i;
    const char *cmdline;
    struct bootmodule *xen_bootmodule;
    struct domain *d;
    int rc;

    dcache_line_bytes = read_dcache_line_bytes();

    percpu_init_areas();
    set_processor_id(0); /* needed early, for smp_processor_id() */

    setup_virtual_regions(NULL, NULL);
    /* Initialize traps early allow us to get backtrace when an error occurred */
    init_traps();

    setup_pagetables(boot_phys_offset);
```

NÃO QUEREMOS PRECISAR EDITAR ESSE TIPO DE COISA >;(

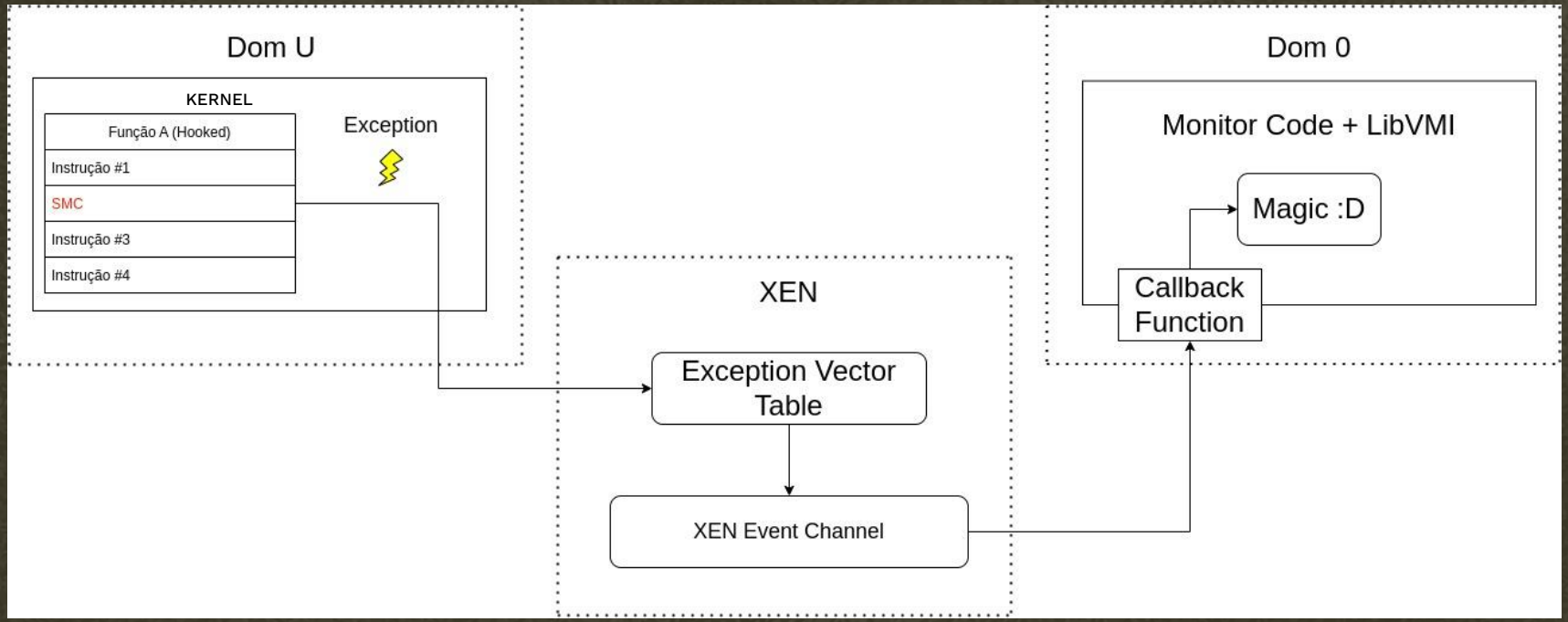
XEN EVENT CHANNELS

- No XEN, as *interrupts/exceptions* são publicadas nos chamados ***"Event Channels"***
  - Um "*Event Channel*" pode ser lido pelo "*Dom0*";
  - Dom0 pode registrar funções de callback:
    - Respondem quando um *interrupt/exception* é publicado.

NÃO PRECISAMOS LIDAR DIRETAMENTE COM A EXCEPTION VECTOR TABLE !!!



FINAL PIPELINE



Elaboração própria.


STEALTH

XEN ALTP2M

VTTBR, Virtualization Translation Table Base Register

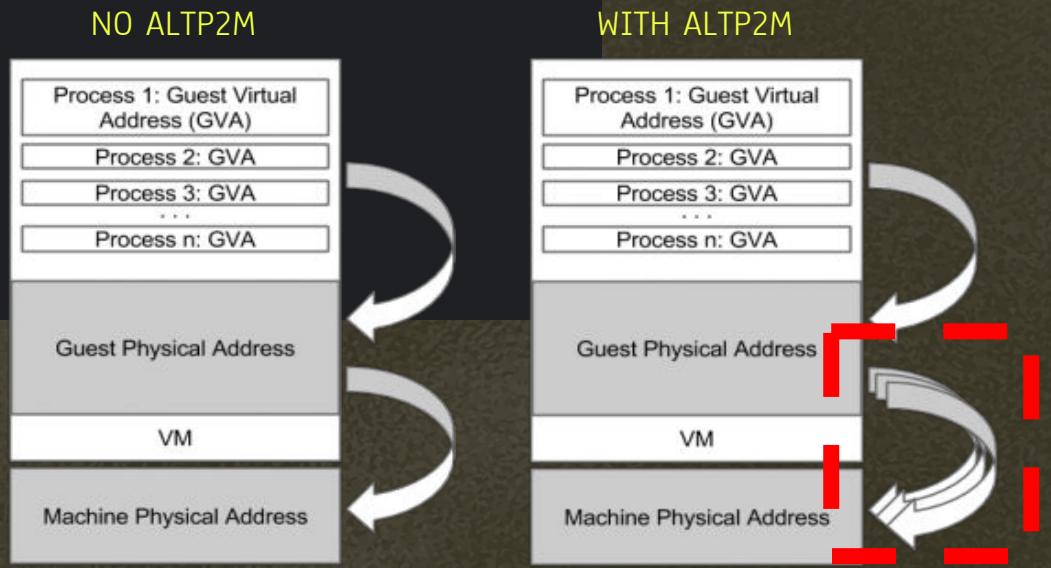
VTTBR holds the base address of the translation table for the stage 2 translation of memory accesses from Non-secure modes other than Hyp mode.

Bit field descriptions

VTTBR is a 64-bit register, and is part of:

- The Virtualization registers functional group.
- The Virtual memory control registers functional group.

Figure B1-78 VTTBR bit assignments



M3M0RY V13W SW1TCH

STEP #1

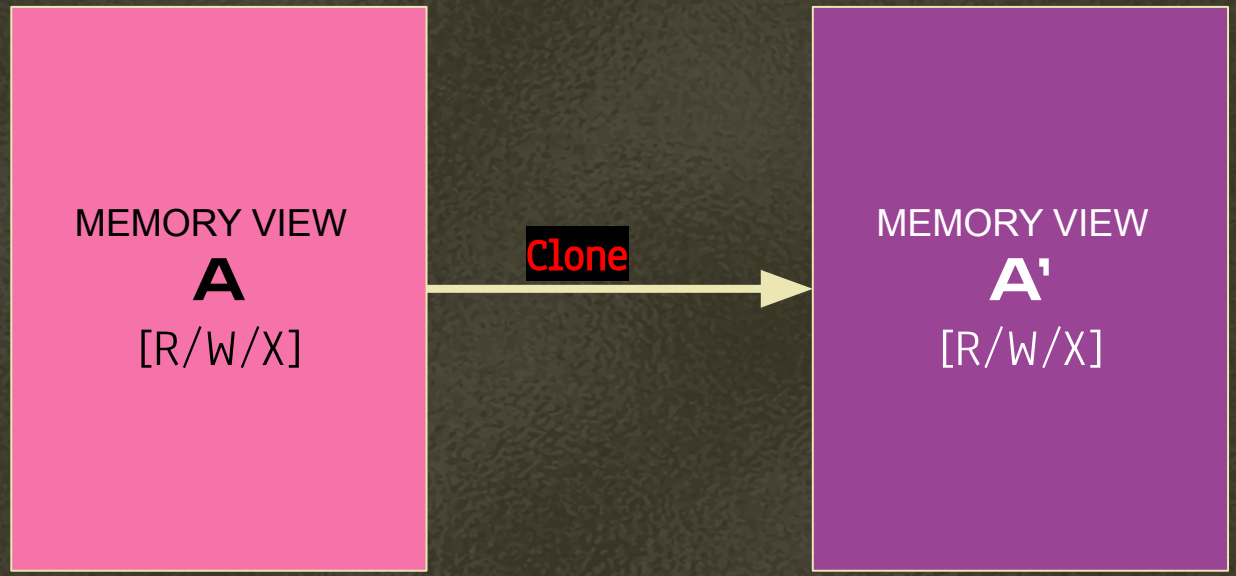
MEMORY VIEW

A

[R/W/X]

M3M0RY V13W SW1TCH

STEP #2



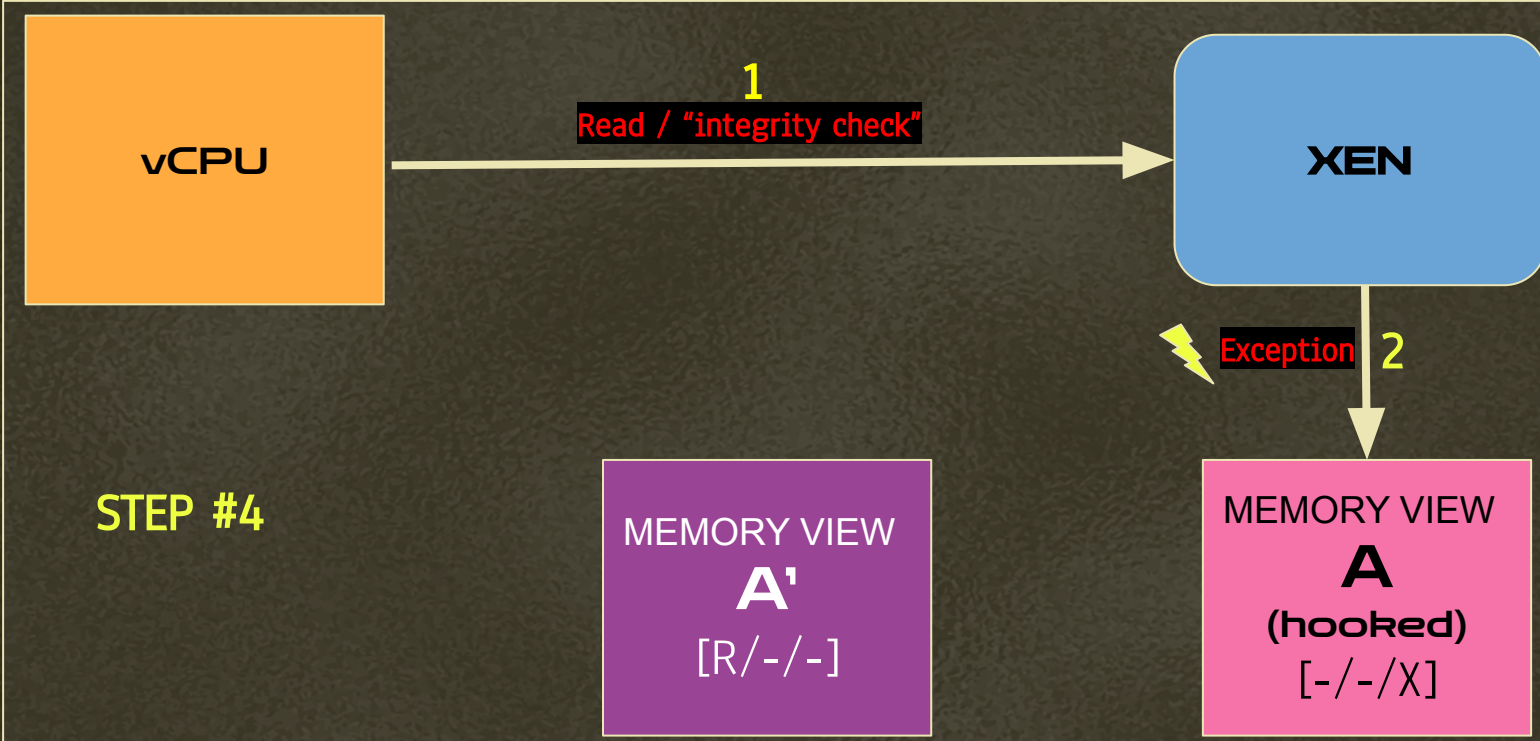
M3M0RY V13W SW1TCH

STEP #3

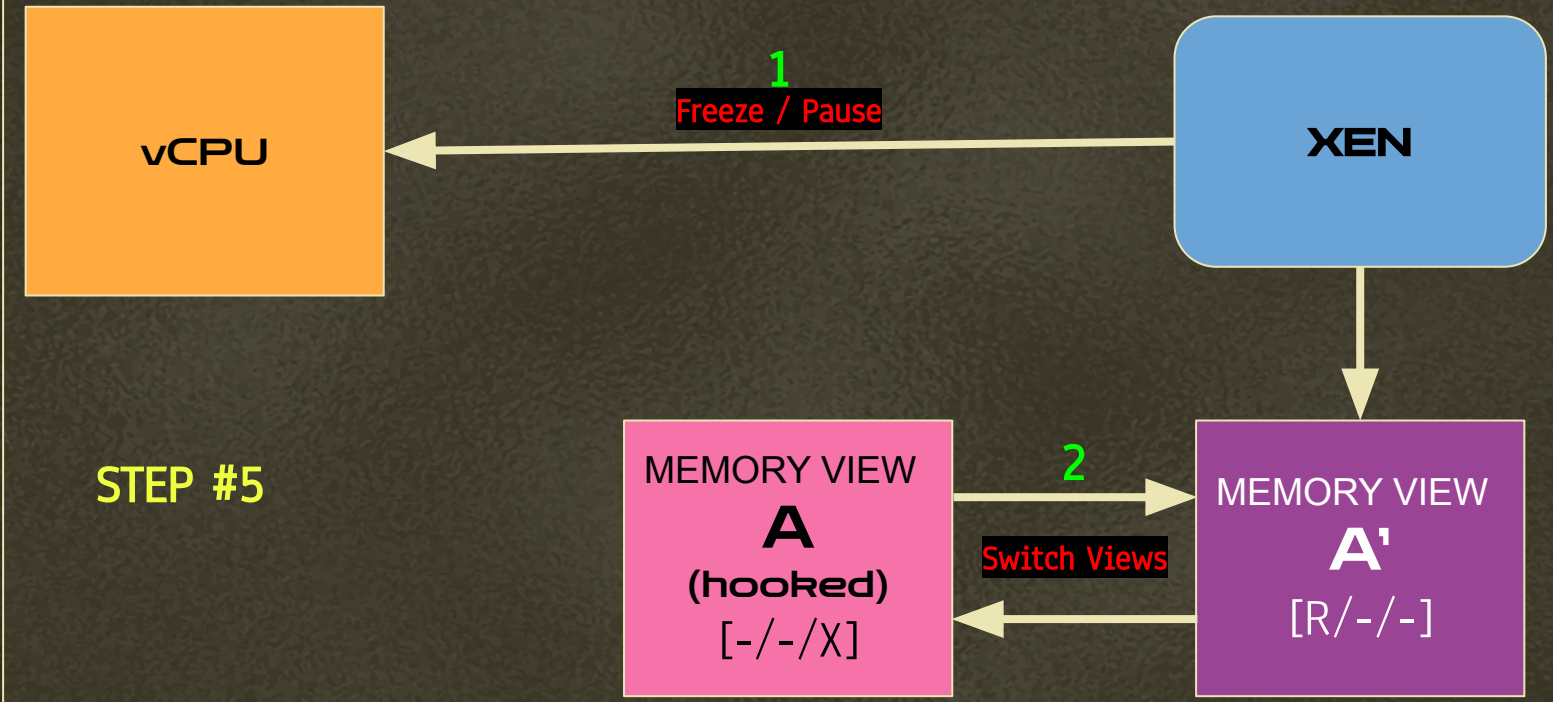
MEMORY VIEW
A
[-/-/X]
(hooked)

MEMORY VIEW
A'
[R/-/-]

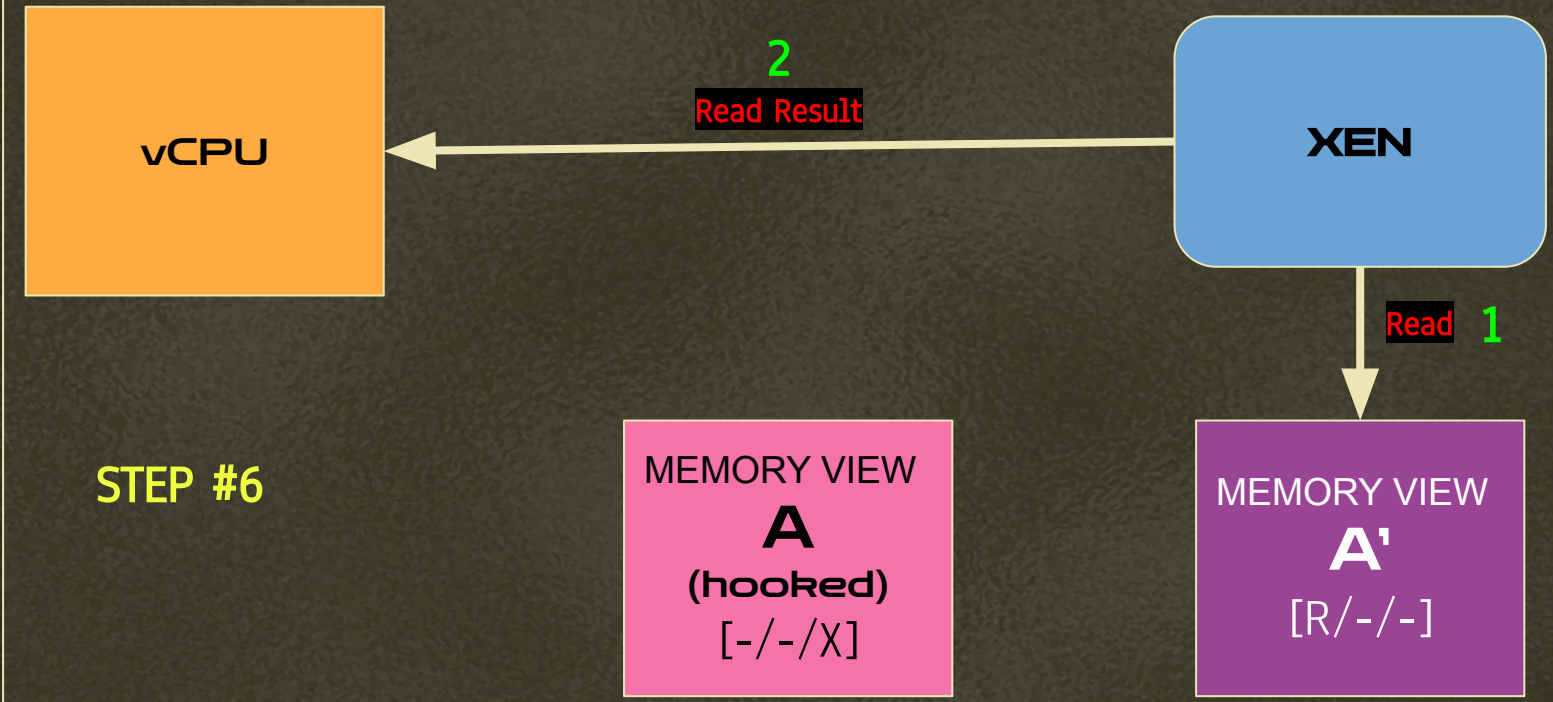
M3M0RY V13W SW1TCH



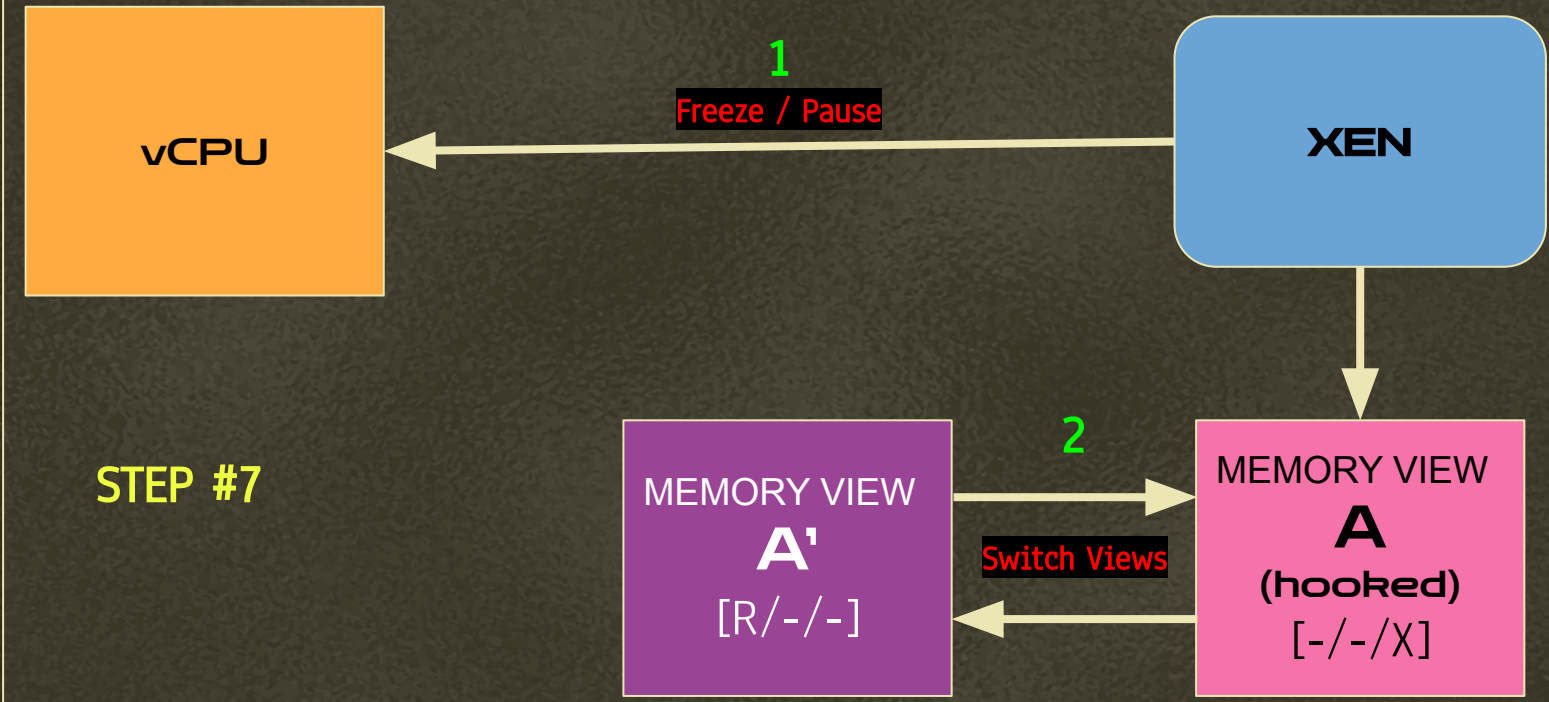
M3M0RY V13W SW1TCH



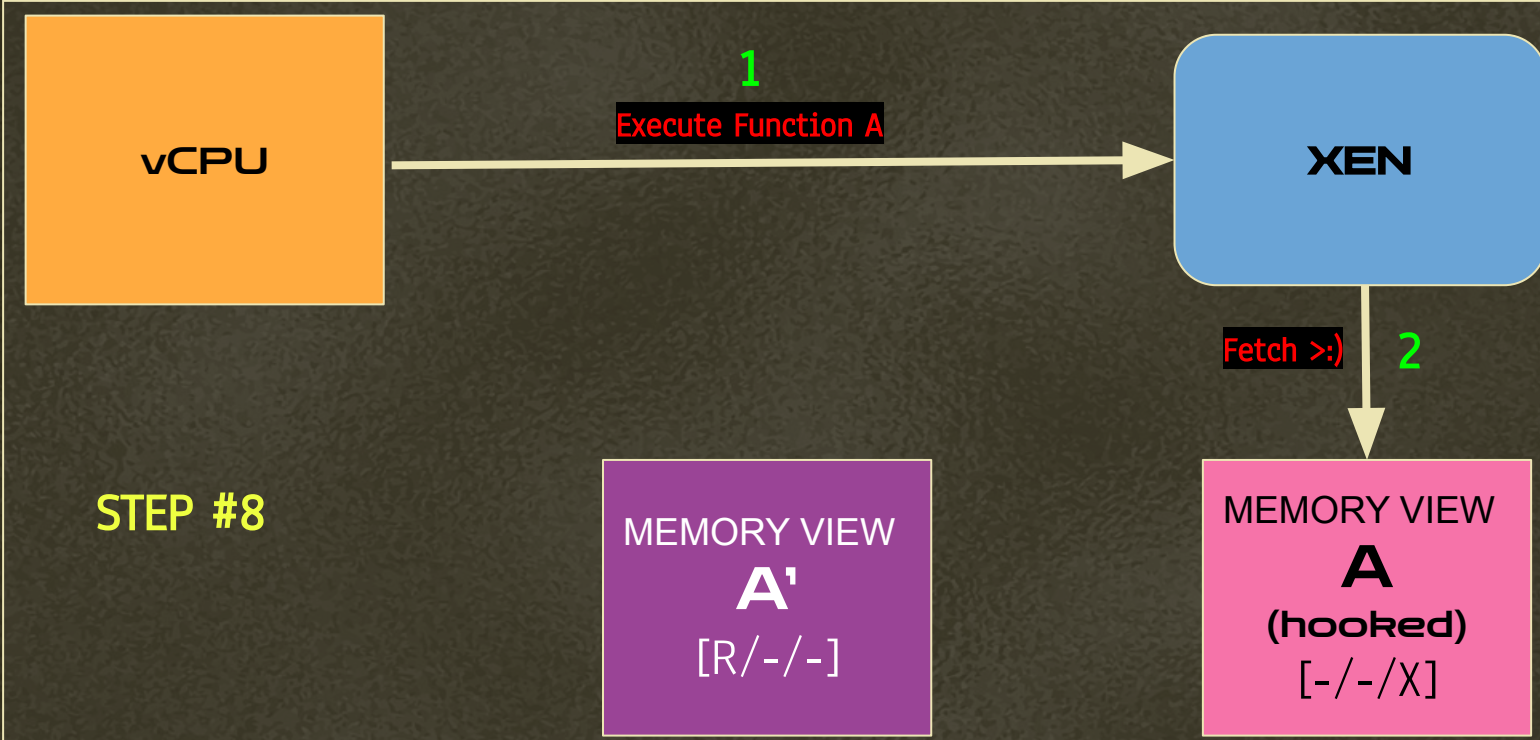
M3M0RY V13W SW1TCH



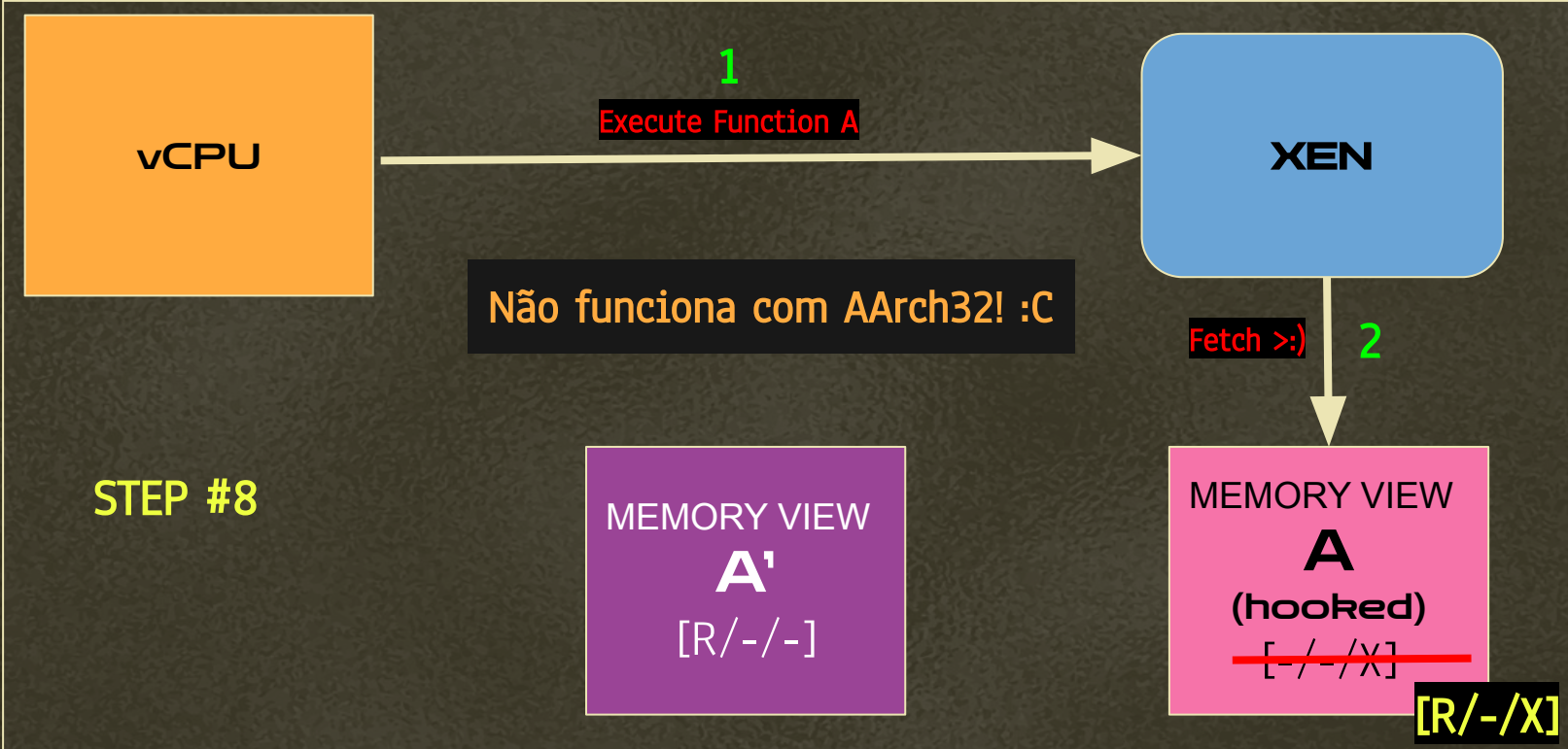
M3M0RY V13W SW1TCH



M3M0RY V13W SW1TCH



M3M0RY V13W SW1TCH



TLBs Split / Cache Incoherence

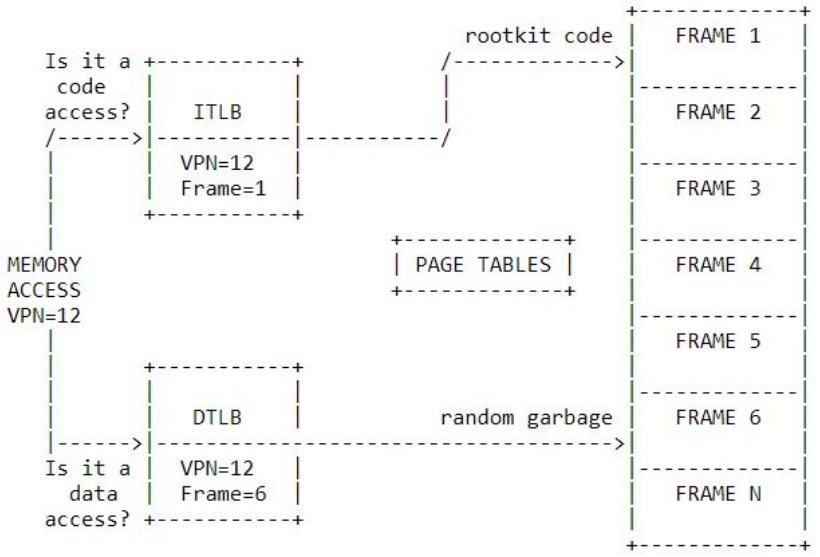
Current issue : #63 | Release date : 2005-01-08 | Editor : Phrack Staff

==Phrack Inc.==

Volume 0x0b, Issue 0x3d, Phile #0x08 of 0x14

|=-------------------------=[Shadow Walker]=-----------------------------=|
|=--------=[Raising The Bar For Windows Rootkit Detection]=-------------=|
|=---=|
|=----------=[Sherri Sparks <ssparks at mail.cs.ucf dot edu >]=----------=|
|=----------=[Jamie Butler <james.butler at hbgary dot com >]=----------=|

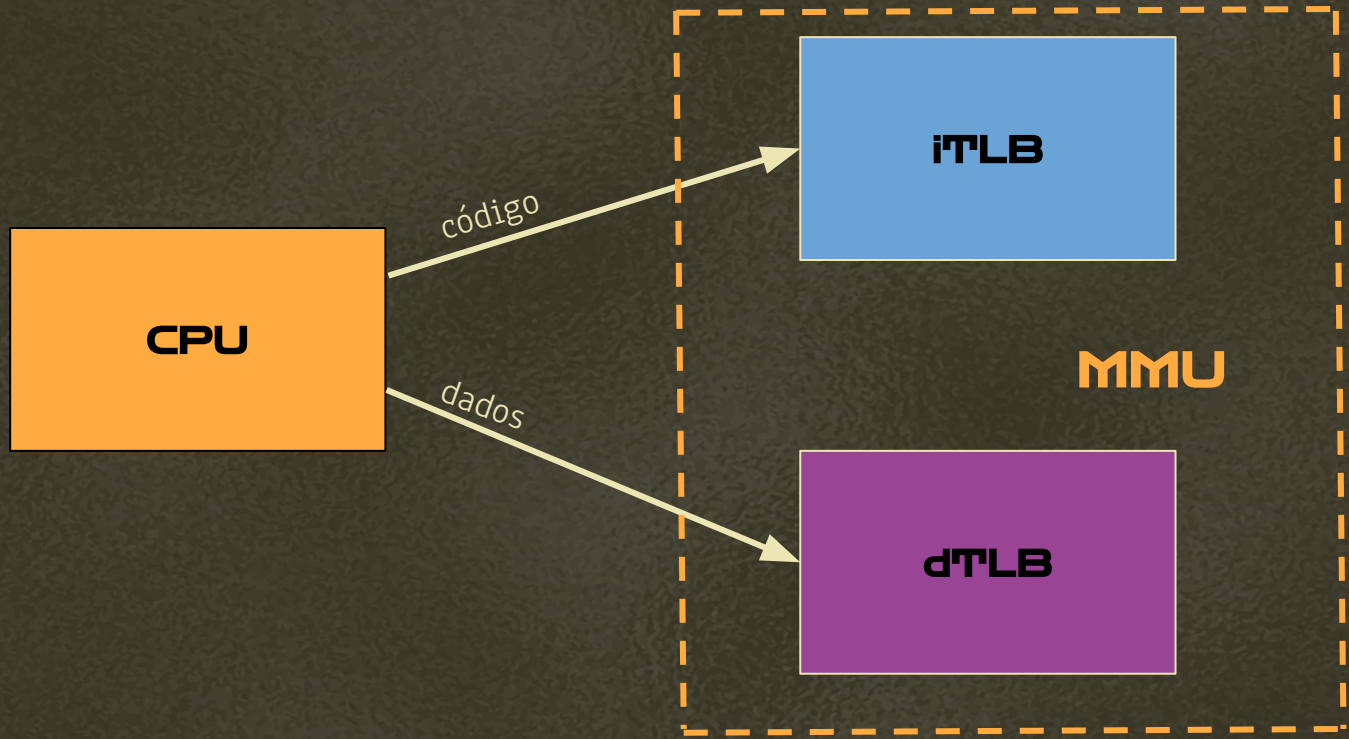
(x86 LAND)



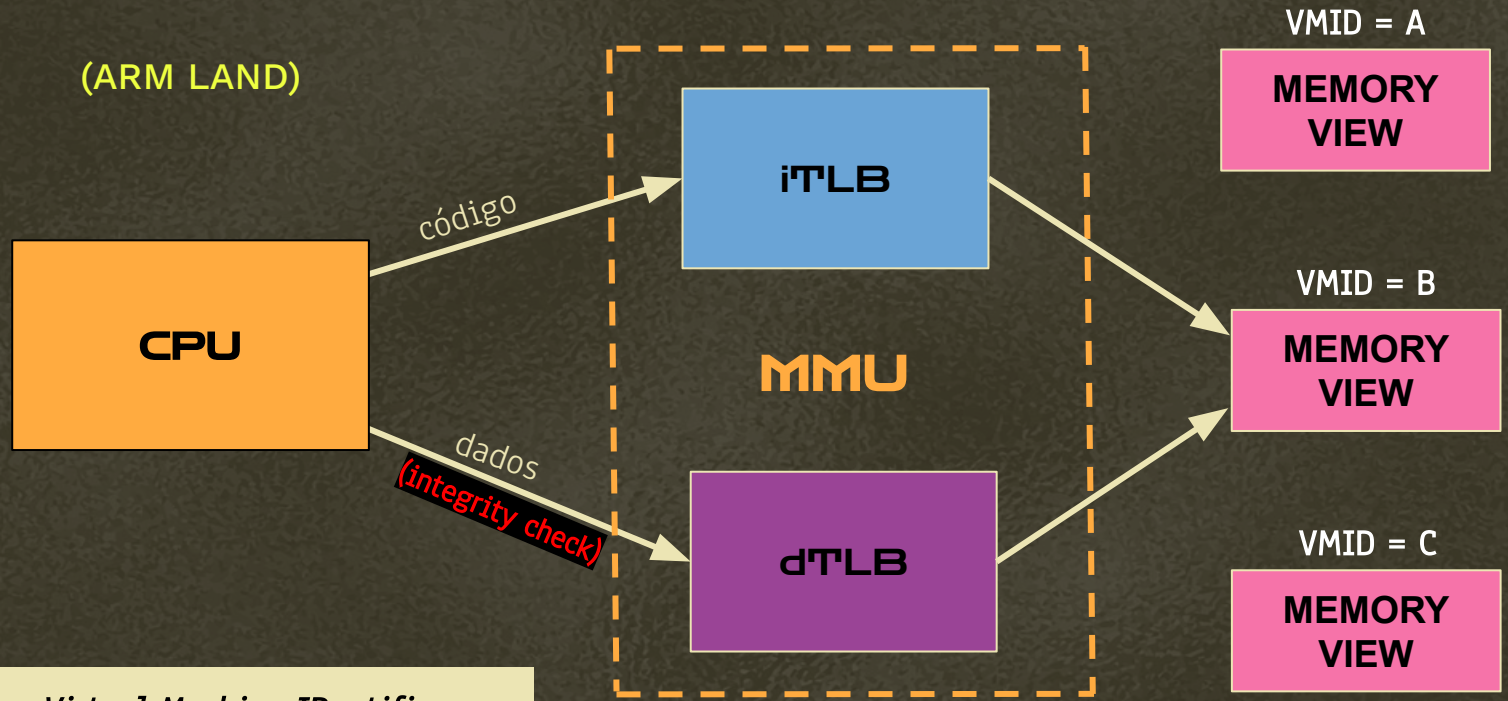
[Figure 5 - Faking Read / Writes by Desynchronizing the Split TLB]

Disponível em: http://phrack.org/issues/63/8.html

TLBs Split / Cache Incoherence

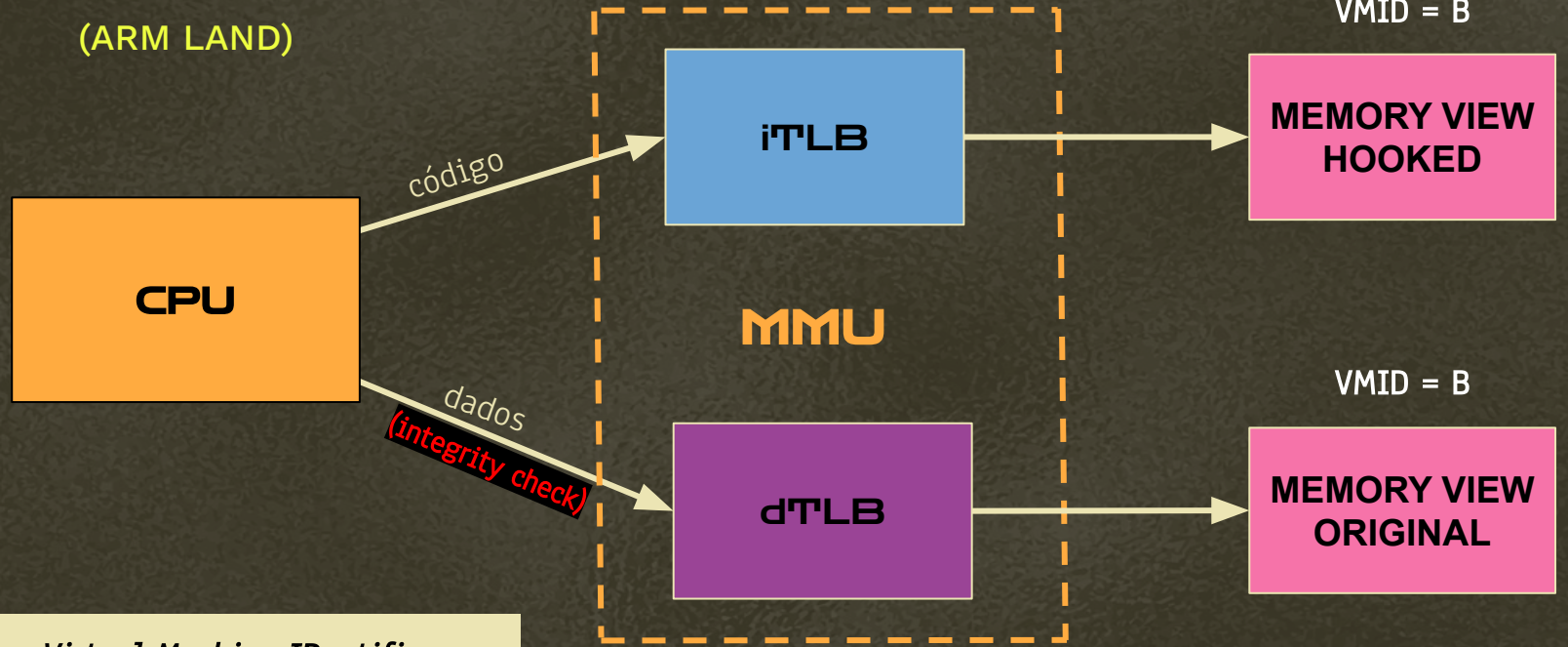


TLBs Split / Cache Incoherence



VMID = *Virtual Machine IDentifier*

TLBs Split / Cache Incoherence

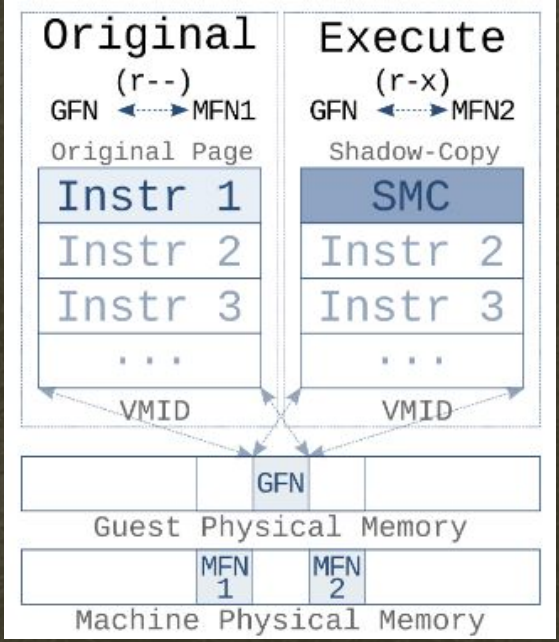


VMID = *Virtual Machine IDentifier*

HIDING IN THE SHADOWS

1. Na primeira execução da função a ser monitorada a busca da instrução (*fetch*) viola as permissões da *memory view* original (*read-only*).
    a. Fazendo com que nenhum mapa de tradução seja salvo nas TLBs.
2. O *Hypervisor* então procede com a interceptação da VM, substituindo a *memory view* original (*read-memory view*), por uma com permissões de execução (*execute-memory view*)
    a. Contendo a *trapping instruction* (SMC) na posição de interesse.

Duas *memory views* dividem o mesmo VMID!



https://dl.acm.org/doi/10.1145/3274694.3274698

HIDING IN THE SHADOWS

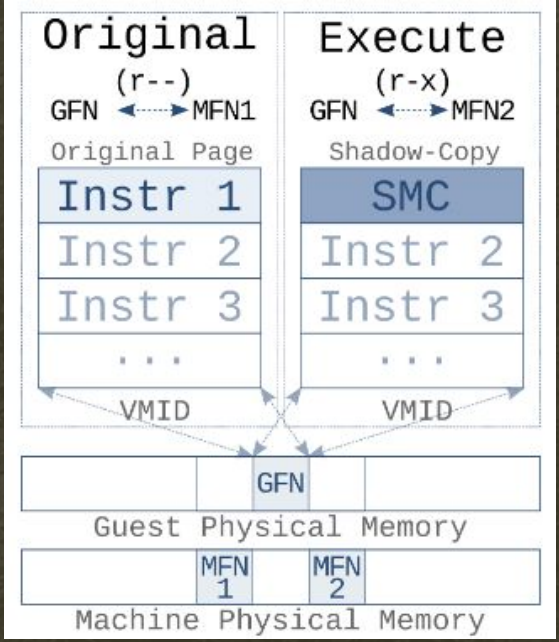
1. Retomando a execução da VM, a tradução (via SLAT) é realizada com sucesso.

2. O mecanismo de tradução adiciona uma entrada na iTLB contendo o endereço físico que é associado a *execute-memory view*.

    a. Consequentemente, as próximas rotinas de tradução vão consultar primeiro a iTLB (não realizando a tradução via SLAT).

3. Após executar a função monitorada, o *Hypervisor* alterna novamente para *memory view* original

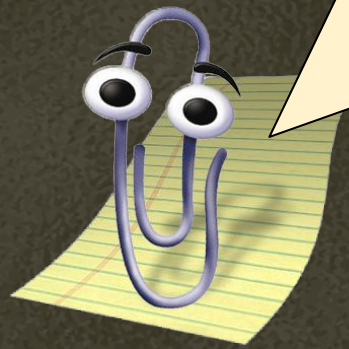
Duas *memory views* dividem o mesmo VMID!



https://dl.acm.org/doi/10.1145/3274694.3274698

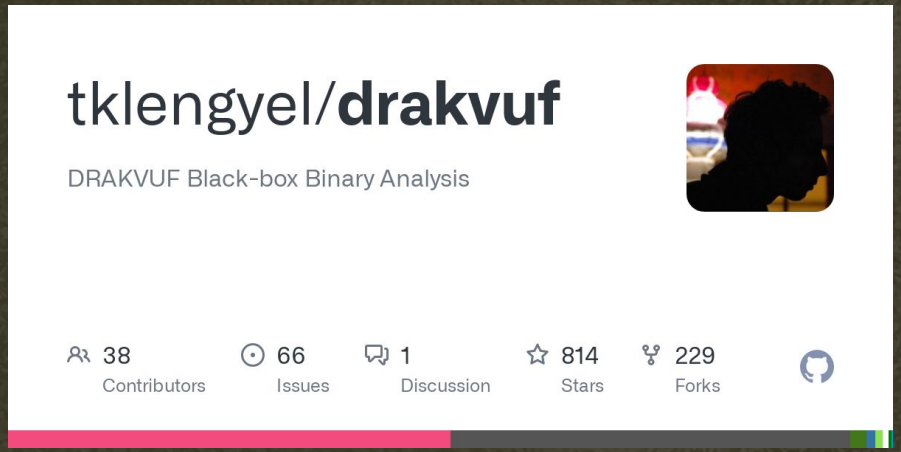
CONCLUSÃO

ALTP2M + CACHE INCOHERENCE = STEALTH SYSCALL HOOKING



DRAKVUF

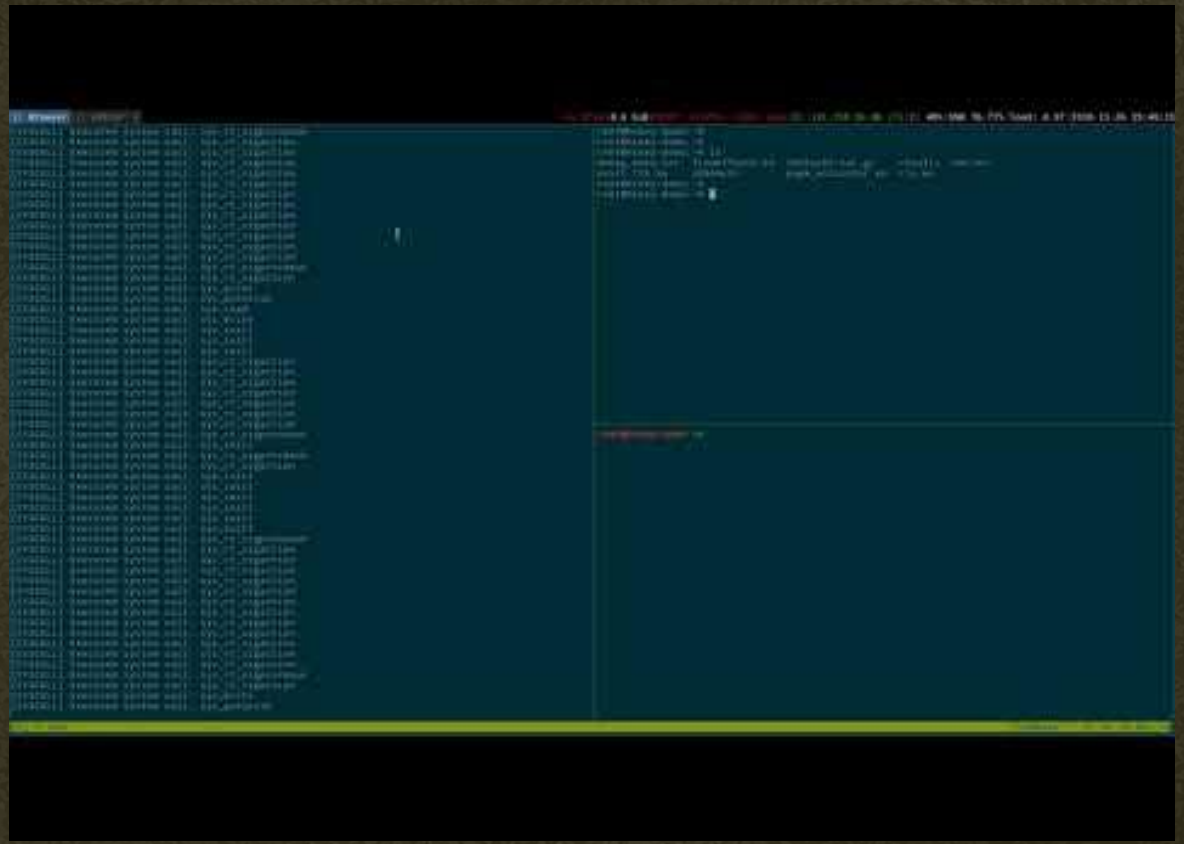
Sergej Proskurin, Tamás K. Lengyel – Stealthy, Hypervisor-based Malware Analysis



https://github.com/tklengyel/drakvuf

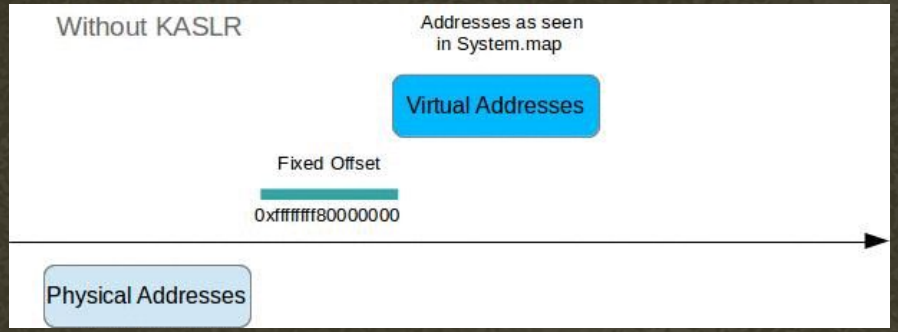
https://www.youtube.com/watch?v=86EvJK2Ef_U

https://github.com/tklengyel/drakvuf/pull/445/commits/d340292a02aa4f775666e9de89f524efdb179dbb



EQ FALTOU

KASLR

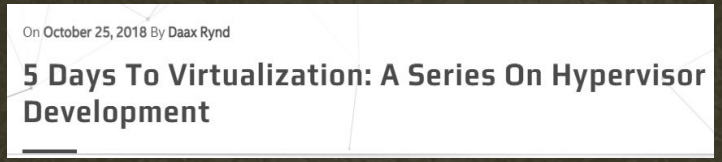


bneuburg.github.io/volatility/kaslr/2017/05/05/KASLR2.html

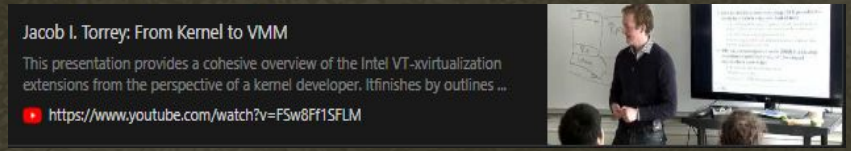
GARANTIR +STEALTH

github.com/a0rtega/pafish

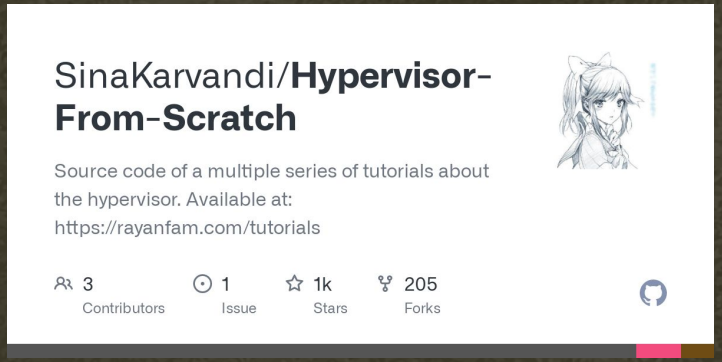
RECOMENDAÇÕES



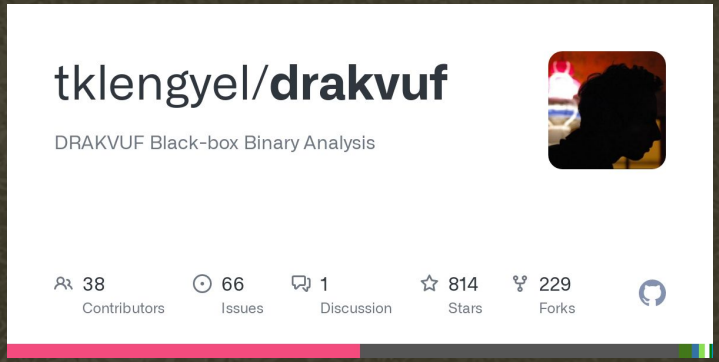
https://revers.engineering/7-days-to-virtualization-a-series-on-hypervisor-development/



https://www.youtube.com/watch?v=FSw8Ff1SFLM



https://github.com/SinaKarvandi/Hypervisor-From-Scratch/



https://github.com/tklengyel/drakvuf

REFERÊNCIAS

ARM (org.). **Learn the architecture - AArch64 Virtualization:** Stage 2 translation. [S. l.], 2019a. Disponível em: https://developer.arm.com/documentation/102142/0100/Stage-2-translation. Acesso em: 7 maio 2022.

ARM (org.). **Learn the architecture - AArch64 Virtualization:** Virtualization in AArch64. [S. l.], 2019b. Disponível em: https://developer.arm.com/documentation/102142/0100/Virtualization-in-AArch64. Acesso em: 7 maio 2022.

ARM (org.). **AArch64 Exception and Interrupt Handling:** AArch64 exception vector table. [S. l.], 2019c. Disponível em: https://developer.arm.com/documentation/100933/0100/AArch64-exception-vector-table. Acesso em: 7 maio 2022.

ARM (org.). **Learn the architecture - AArch64 Virtualization:** Secure virtualization. [S. l.], 2019d. Disponível em: https://developer.arm.com/documentation/102142/0100/Secure-virtualization. Acesso em: 7 maio 2022.

DE BOCK, Yorick; MERCELIS, Siegfried; BROECKHOVE, Jan; et al. Real-time virtualization with Xvisor. **Internet of Things**, v. 11, p. 100238, 2020. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S2542660520300718>. Acesso em: 16 jun. 2022.

REFERÊNCIAS

PROSKURIN, S. et al. **Hiding in the Shadows: Empowering ARM for Stealthy Virtual Machine Introspection**. Proceedings of the 34th Annual Computer Security Applications Conference, 2018. Disponível em: <https://dl.acm.org/doi/10.1145/3274694.3274698>. Acesso em: 30 ago. 2022

SLOSS, Andrew N; SYMES, Dominic; WRIGHT, Chris. **ARM system developer's guide designing and optimizing system software.** Estados Unidos: Elsevier/ Morgan Kaufman, 2004.

TANENBAUM, Andrew S. **Modern operating systems**. Quarta edição. Boston: Pearson, 2015.

XEN PROJECT (org.). **Xen ARM with Virtualization Extensions whitepaper**. [S. l.], 2018a Disponível em: https://wiki.xenproject.org/wiki/Xen_ARM_with_Virtualization_Extensions_whitepaper. Acesso em: 7 maio 2022.

XEN PROJECT (org.). **Xen Project Software Overview**. [S. l.], 2018b. Disponível em: https://wiki.xenproject.org/wiki/Xen_Project_Software_Overview. Acesso em: 7 maio 2022.

! MUITO OBRIGADO !

Q && A

@sargx90

/in/fernando-menardi

sargx#0654