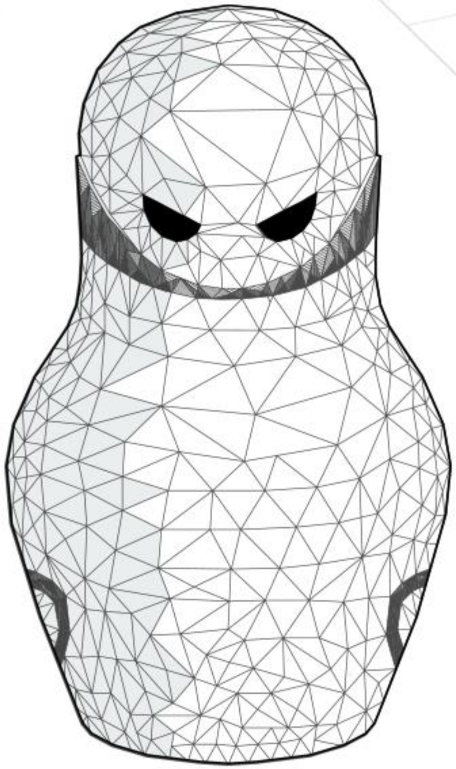


Attacking Hypervisors via Firmware and Hardware

Alex Matrosov (@matrosov),
Mikhail Gorobets,
Oleksandr Bazhaniuk (@ABazhaniuk),
Andrew Furtak,
Yuriy Bulygin (@c7zero)

Intel Security Advanced Threat Research

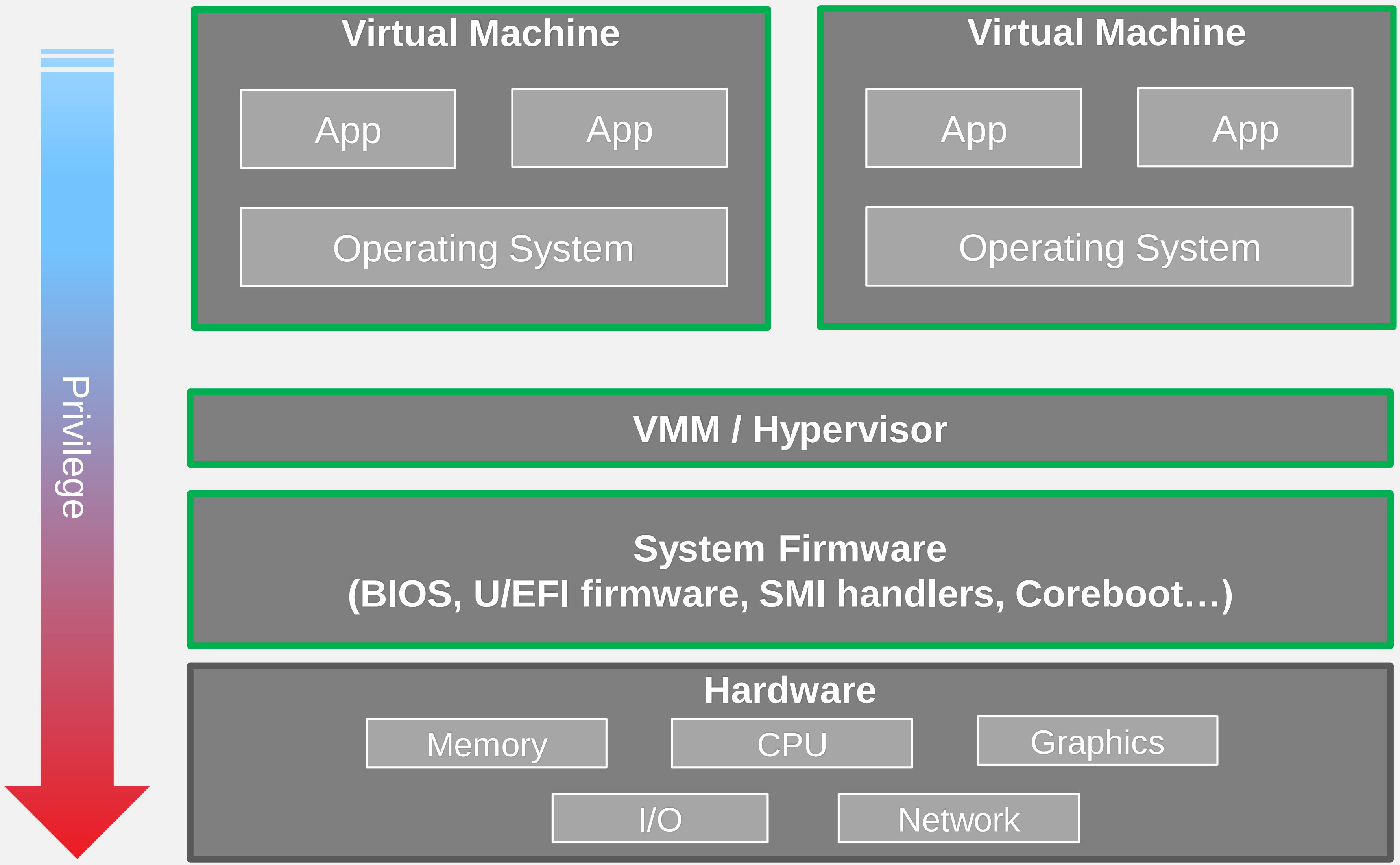
Agenda

- Hypervisor based isolation
- Firmware rootkit vs hypervisor
- Attacking hypervisors through system firmware
- Tools and mitigations
- Conclusions

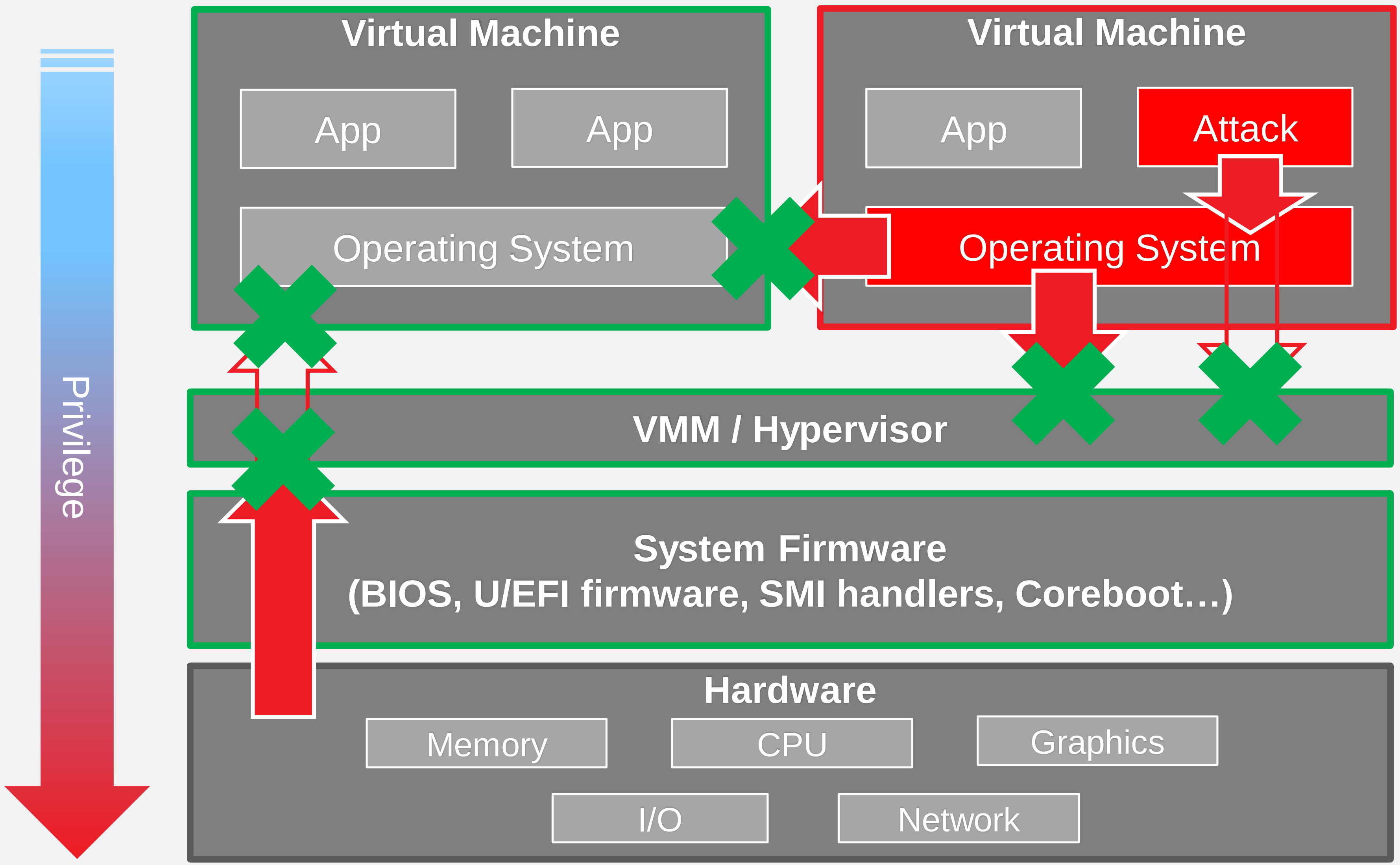
Hypervisor Based Isolation

Image [source]

Hypervisor Based Isolation



Hypervisor Based Isolation



Hypervisor Protections

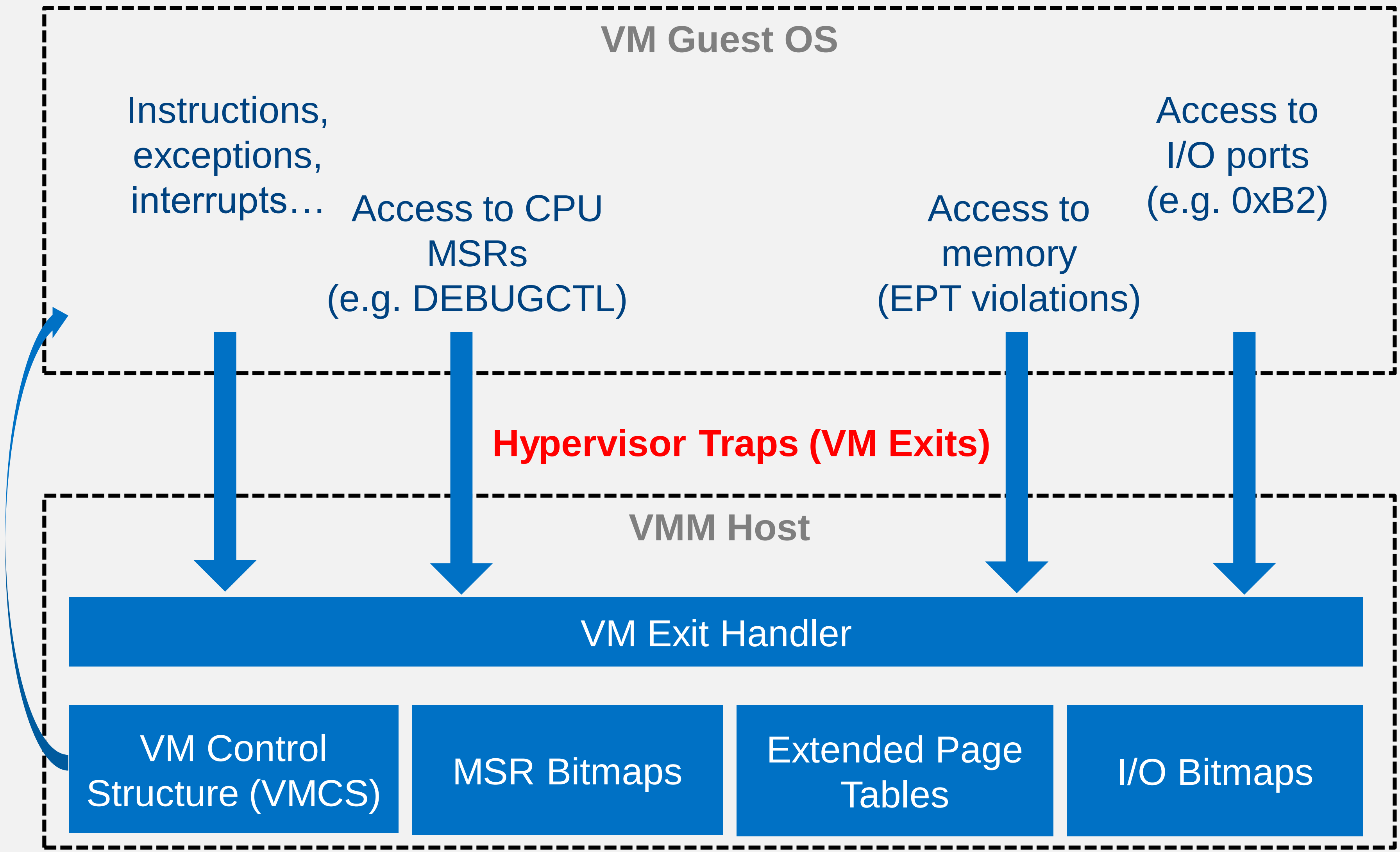
Software Isolation

- **CPU / SoC:** traps to hypervisor (*VM Exits*), MSR & I/O permissions bitmaps, rings (PV)…
- **Memory / MMIO**: hardware page tables (e.g. EPT, NPT), software shadow page tables

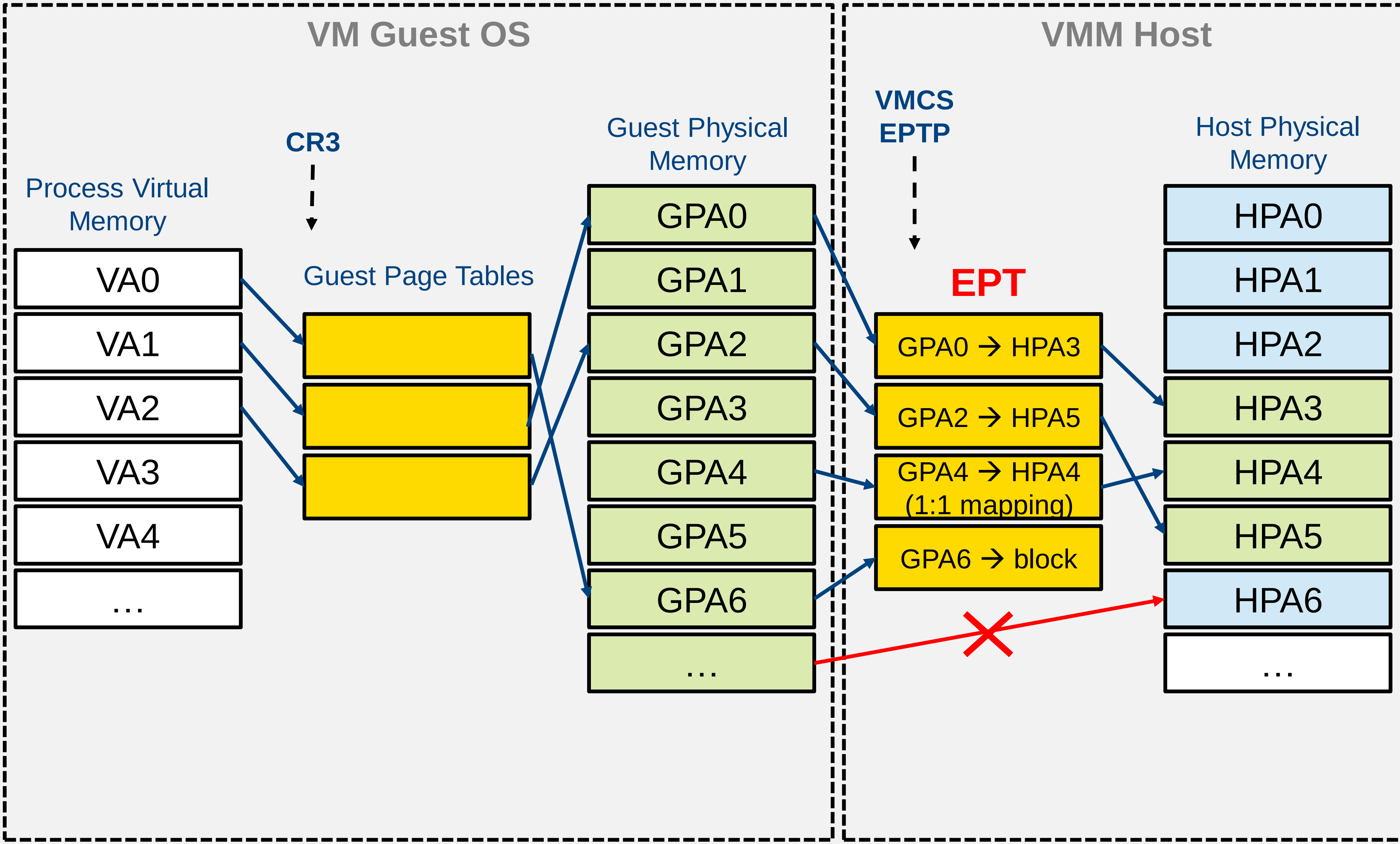
Devices Isolation

- **CPU / SoC:** interrupt remapping
- **Memory / MMIO**: IOMMU, No-DMA ranges

CPU Virtualization (simplified)



Protecting Memory with HW Assisted Paging



Hypervisor Protections

System Firmware Isolation

Firmware Rootkit vs Hypervisor

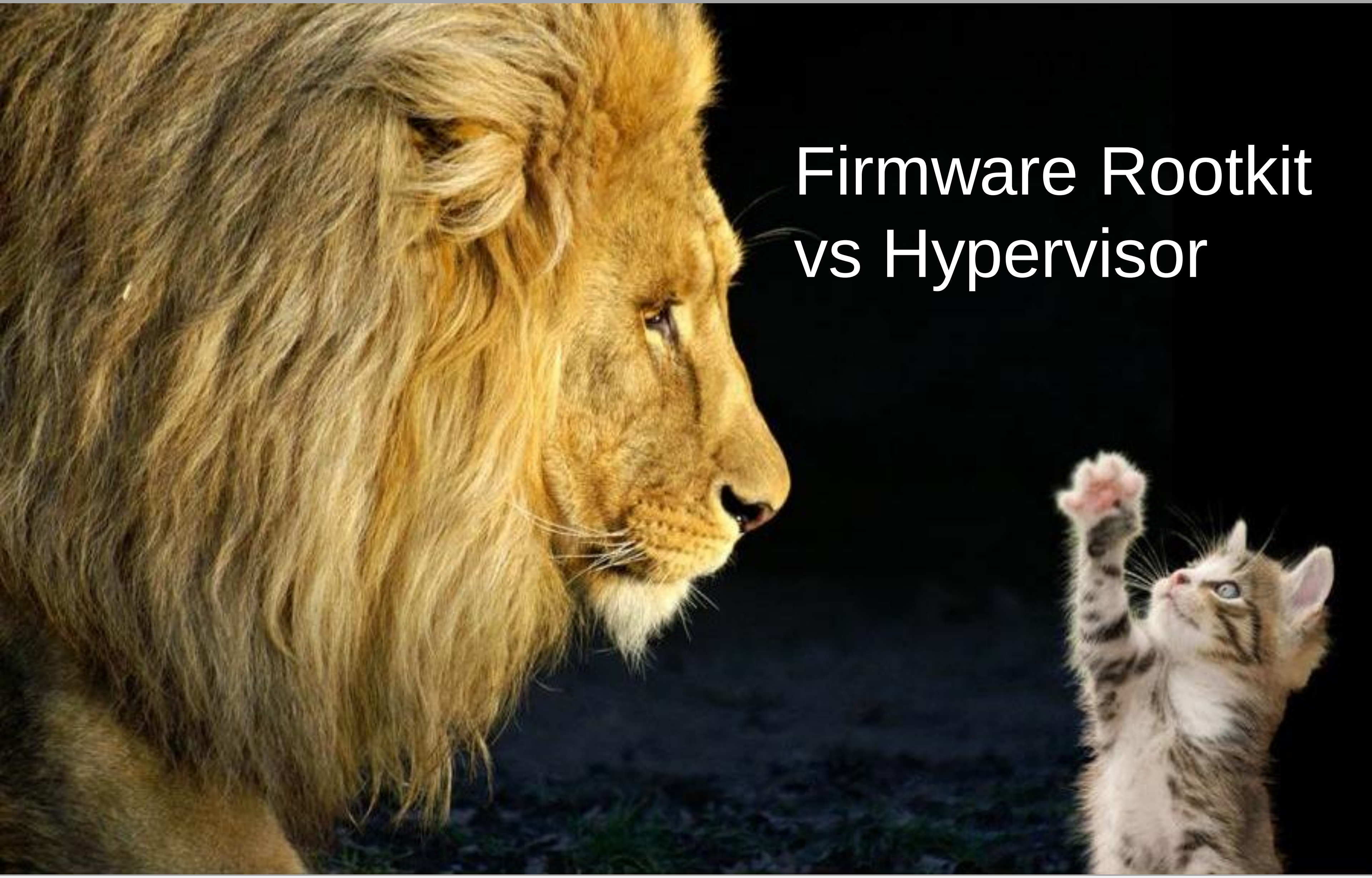
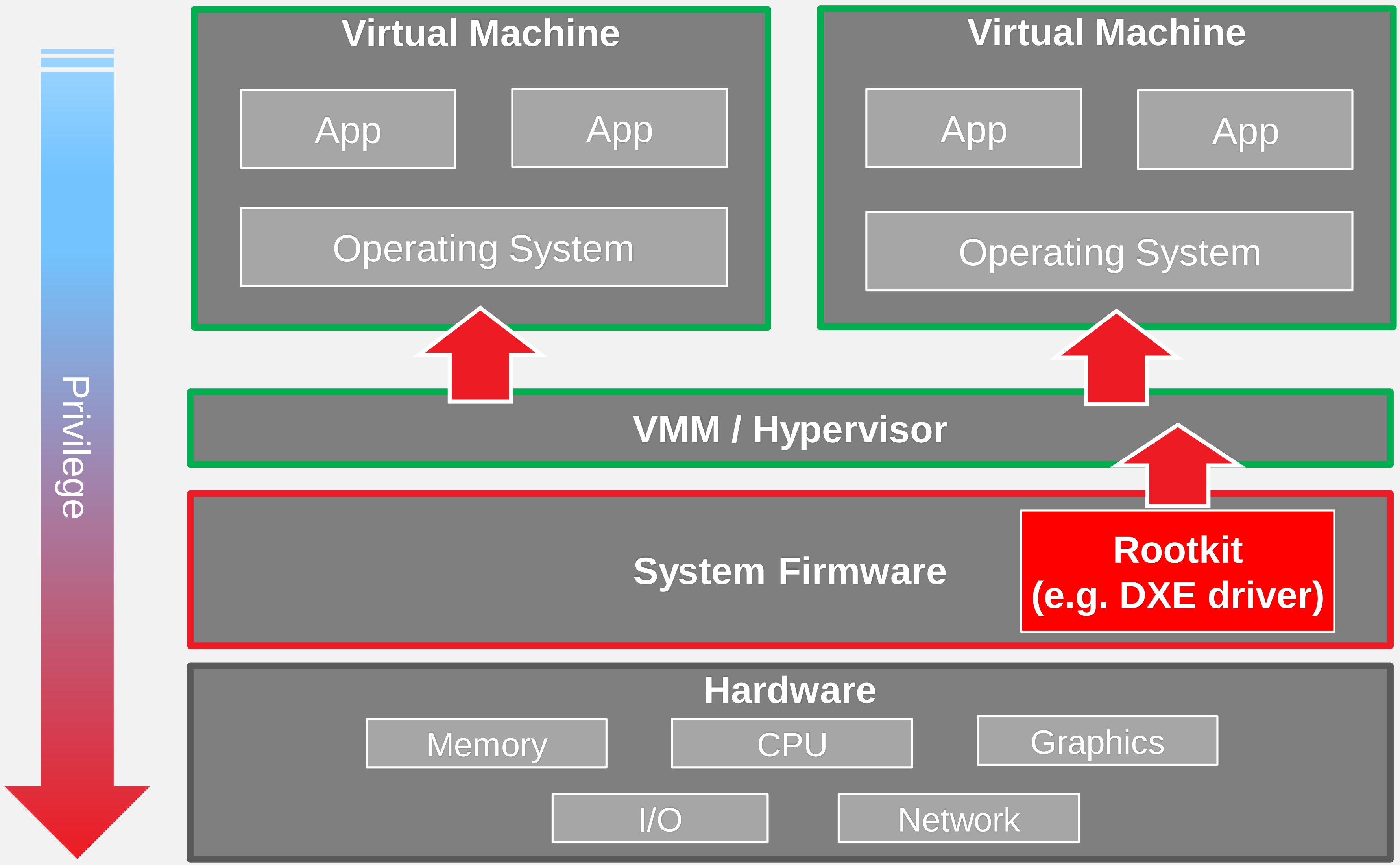
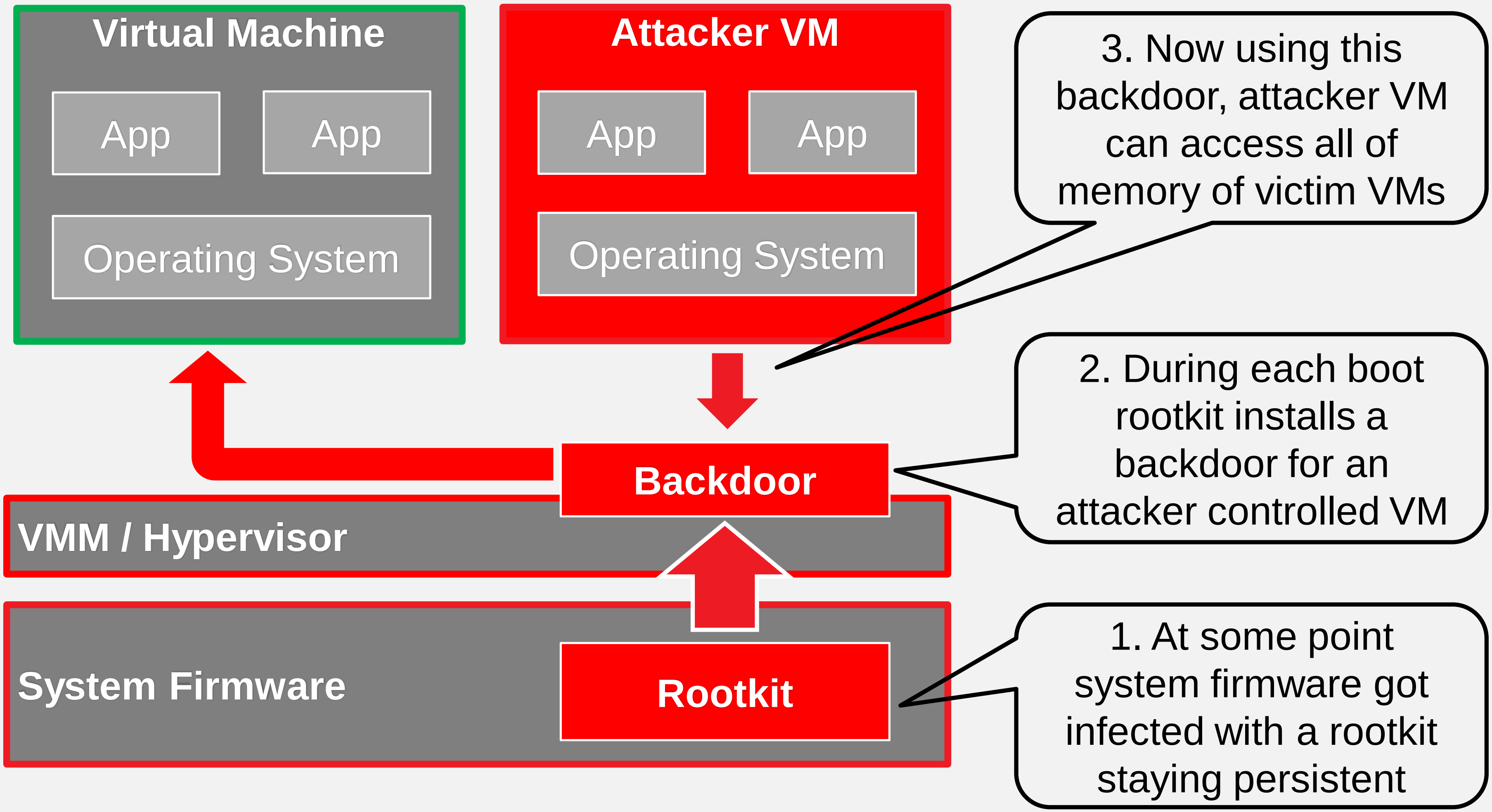


Image source

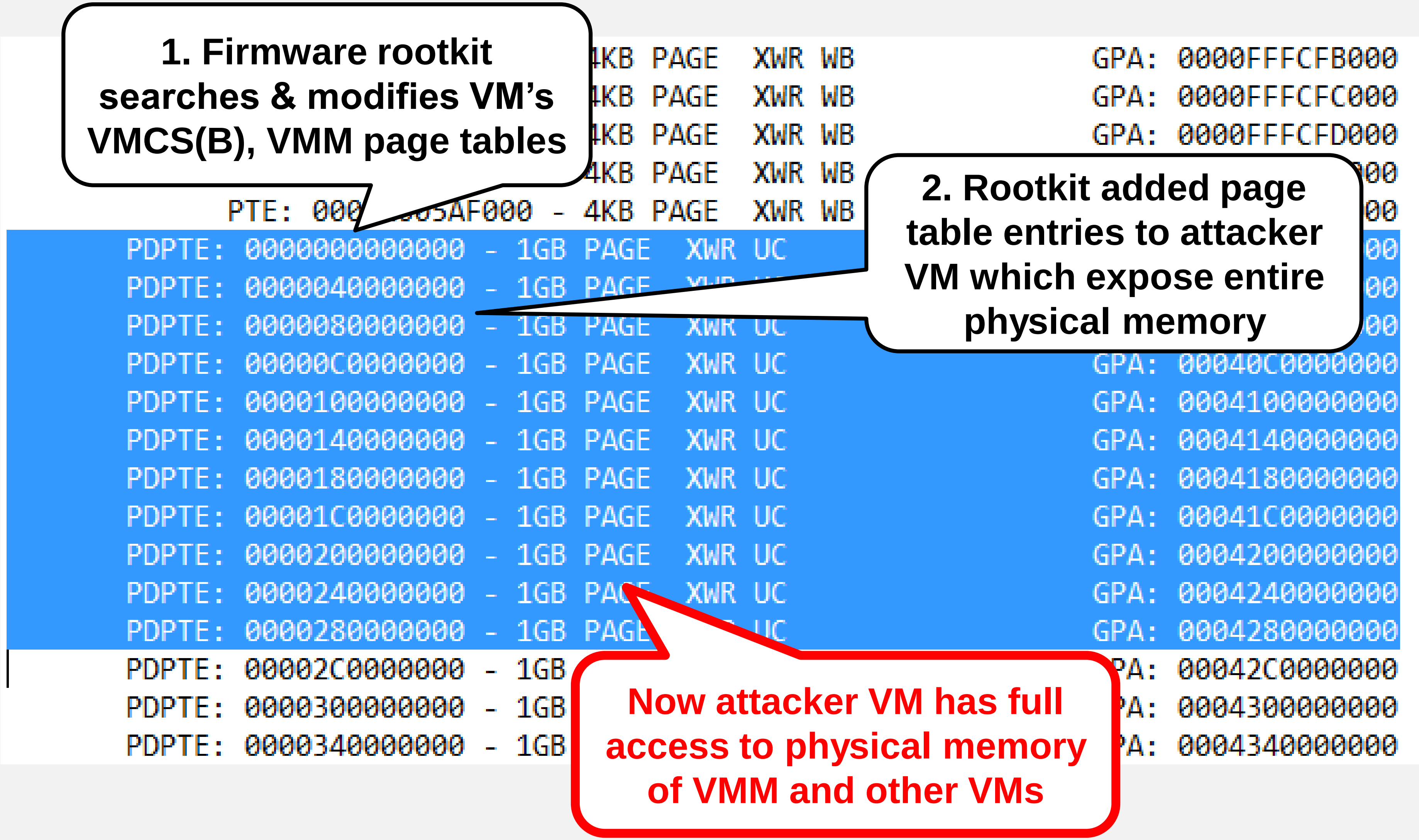
What is firmware rootkit?



Firmware rootkit can open a backdoor for an attacker VM to access all other VMs

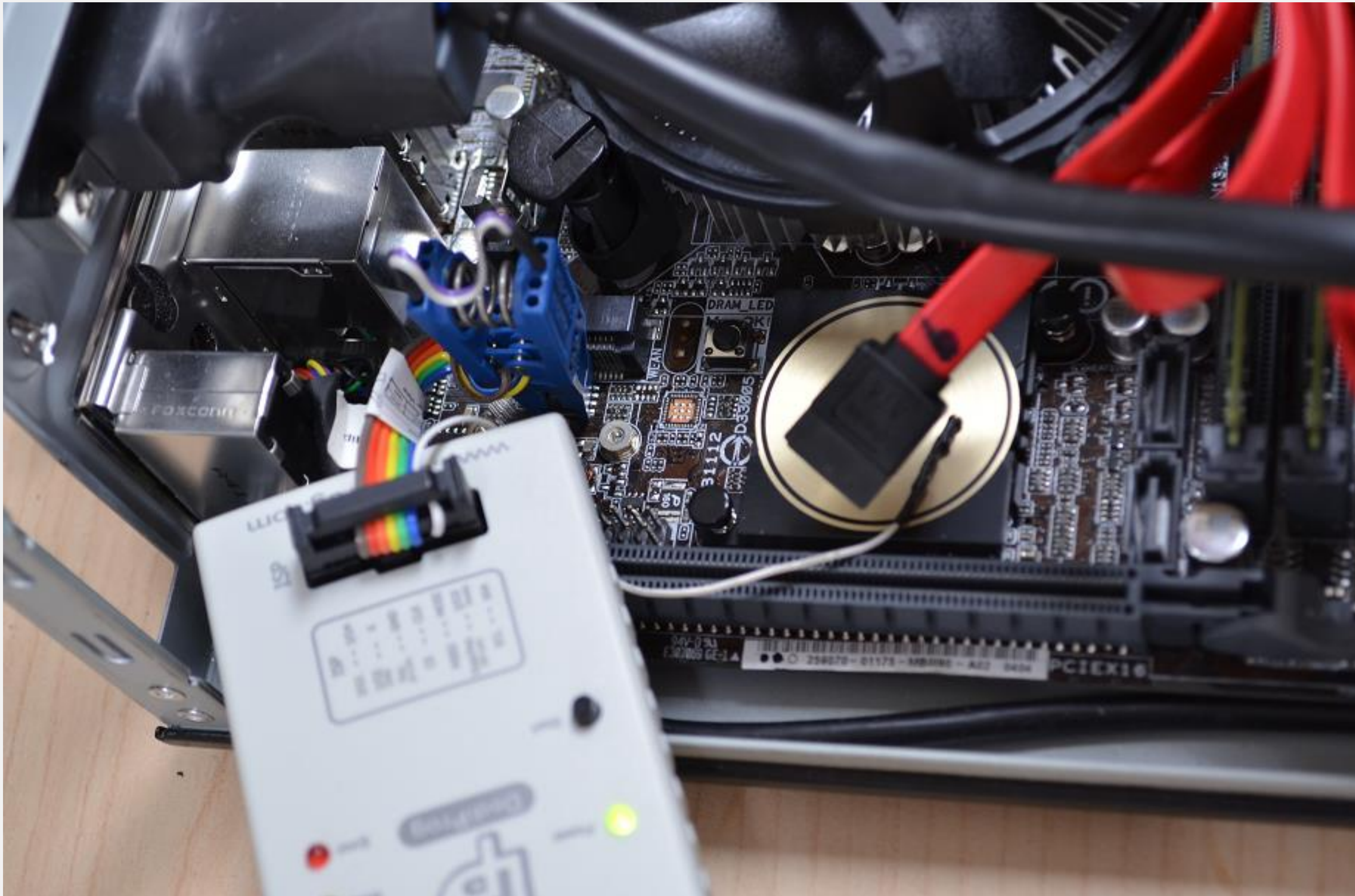


“Backdoor” for attacker’s VM



So how would one install a rootkit in the firmware?

Using hardware SPI flash programmer…

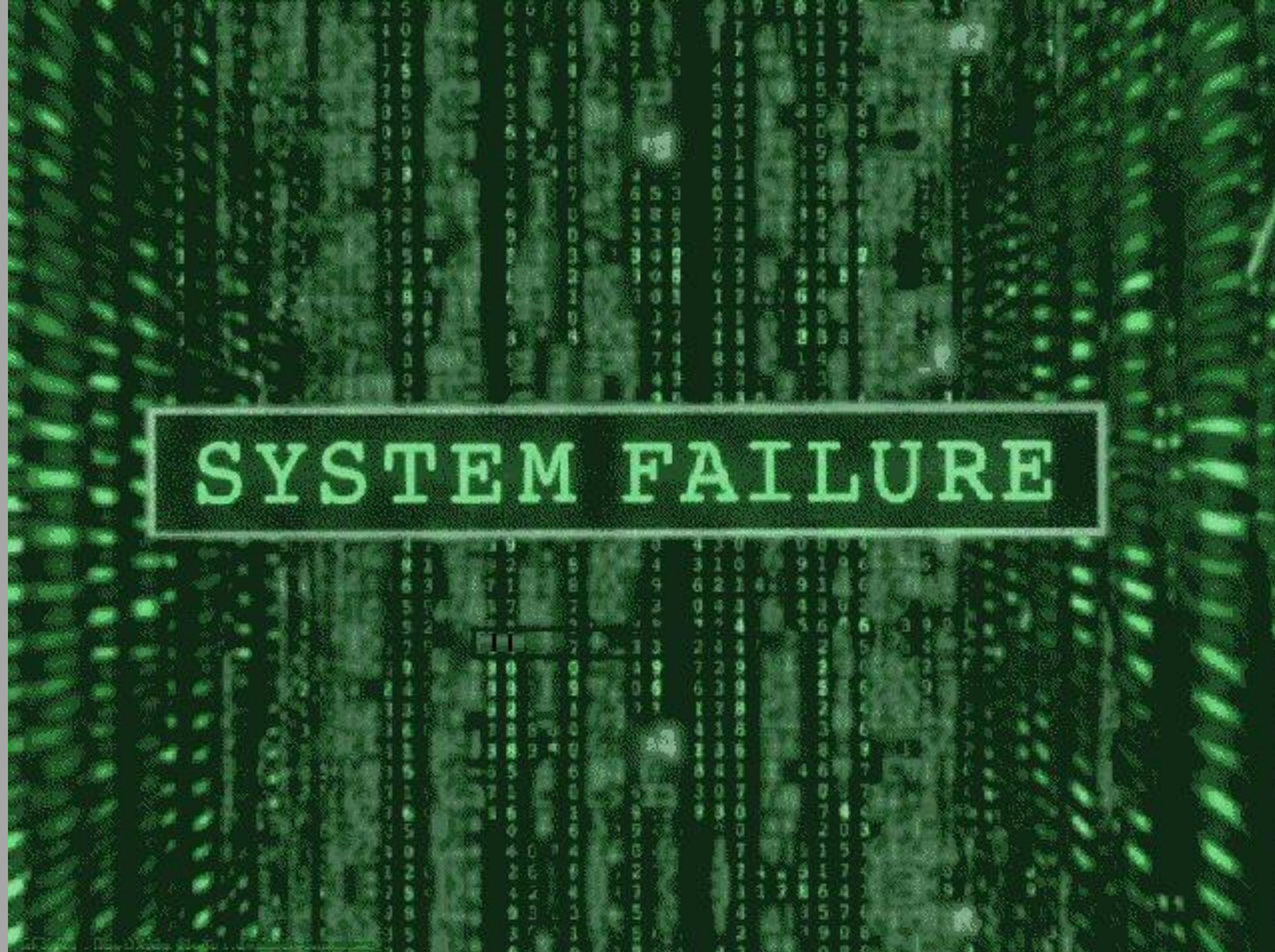


USB & exploiting weak firmware protections...

Software access and exploiting some vulnerability in firmware …

- From privileged guest (e.g. Dom0). Requires privesc from normal guest (e.g. DomU) or remote
- From the host OS before/in parallel to VMM
- From normal guest if firmware is exposed to the guest by VMM

For example, if firmware is not adequately write protected in system flash memory



DEMO

Rootkit in System Firmware Exposes Secrets from Virtual Machines

https://youtu.be/sJnIiPN0104

Image source

- We *flashed* rootkited part of firmware image from within a root partition to install the rootkit
- **The system doesn't properly protect firmware** in SPI flash memory so we could bypass write-protection
- Finally more systems protect firmware on the flash memory

`common.bios_wp`

CHIPSEC module to test write-protection

- Malware can exploit vulnerabilities in firmware to install a rootkit on such systems

Attacking and Defending BIOS in 2015

VMM “forensics”

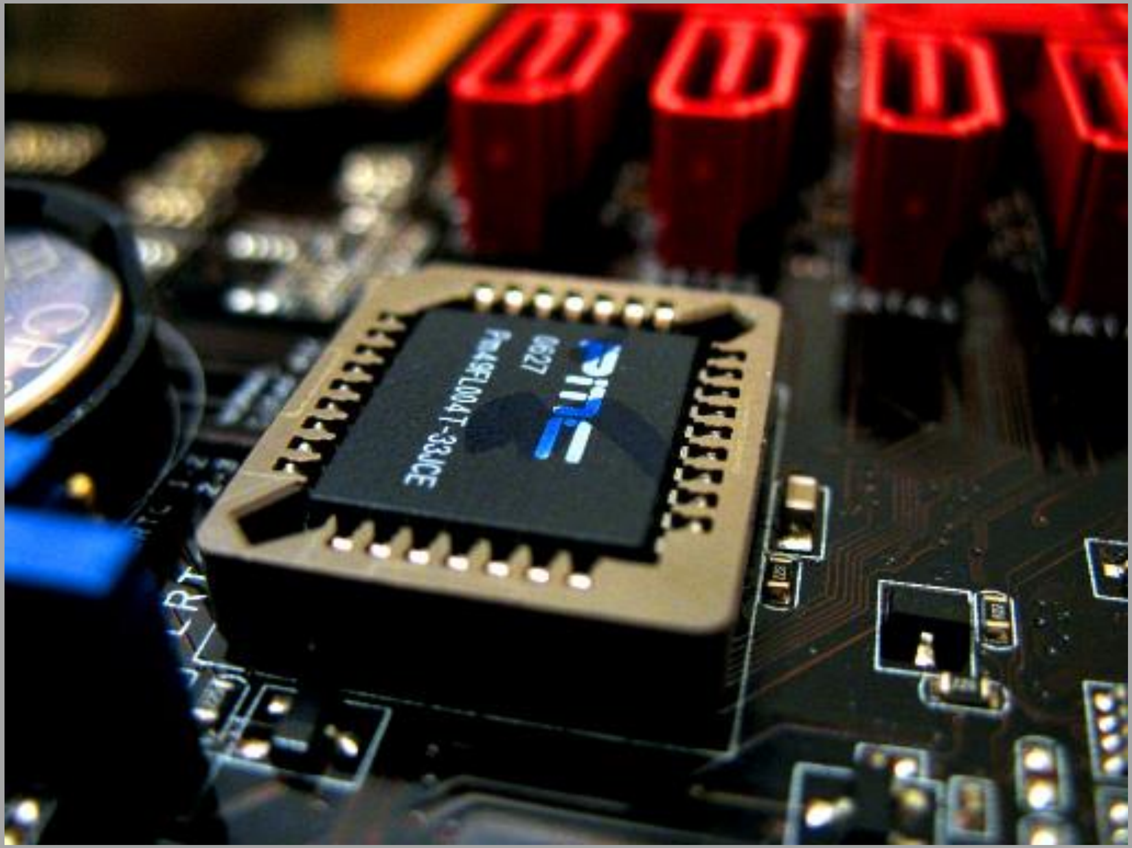
With the help of a rootkit in firmware any VM guest can extract all information about hypervisor and other VMs … and just from memory

- VMCS structures, MSR and I/O bitmaps for each VM guest
- EPT for each VM guest
- Regular page tables for hypervisor and each VM guest
- IOMMU pages tables for each IOMMU device
- Full hypervisor memory map, VM exit handler…
- Real hardware configuration (registers for real PCIe devices, MMIO contents…)

VMM Hardware Page Tables…

```
EPTP: 0x0000004ac8000
  PML4E: 0x0000004b1c000
    PDPTE: 0x0000004b1a000
      PDE   : 0x0000004b13000
        PTE   : 0x0000000000000  - 4KB PAGE  XWR        GPA: 0x0000000000000
        PTE   : 0x0000000002000  - 4KB PAGE  XWR        GPA: 0x0000000002000
        PTE   : 0x0000000003000  - 4KB PAGE  XWR        GPA: 0x0000000003000
        PTE   : 0x0000000004000  - 4KB PAGE  XWR        GPA: 0x0000000004000
        PTE   : 0x0000000005000  - 4KB PAGE  XWR        GPA: 0x0000000005000
        PTE   : 0x0000000006000  - 4KB PAGE  XWR        GPA: 0x0000000006000
```

```
EPT Host physical address ranges:
  0x0000000000000 - 0x0000000000fff       1  XWR
  0x0000000002000 - 0x000000009cfff     155  XWR
  0x00000000c0000 - 0x00000000c7fff       8  XWR
  0x00000000c9000 - 0x00000000c9fff       1  XWR
  0x00000000ce000 - 0x00000000cefff       1  XWR
  0x00000000e0000 - 0x0000000192fff     179  XWR
  0x0000000195000 - 0x0000000195fff       1  --R
  0x0000000196000 - 0x0000000196fff       1  XWR
  0x0000000198000 - 0x0000000199fff       2  XWR
  0x000000019e000 - 0x00000001a3fff       6  XWR
  0x00000001a6000 - 0x00000001c4fff      31  XWR
  0x00000001c8000 - 0x00000001c8fff       1  XWR
  0x00000001cb000 - 0x00000001dcfff      18  XWR
```

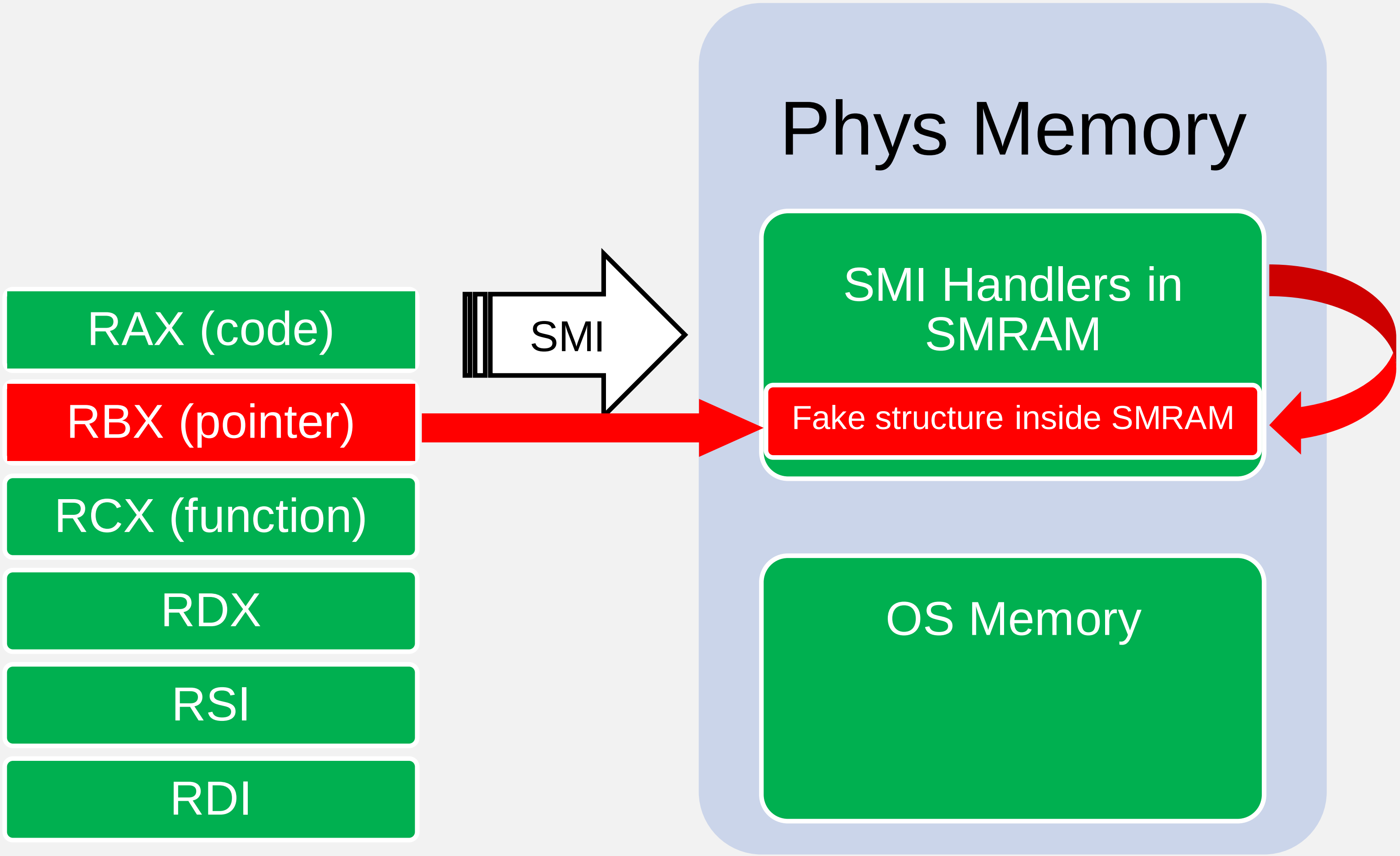


Attacking Hypervisors through System Firmware

(with OS kernel access)

Image source

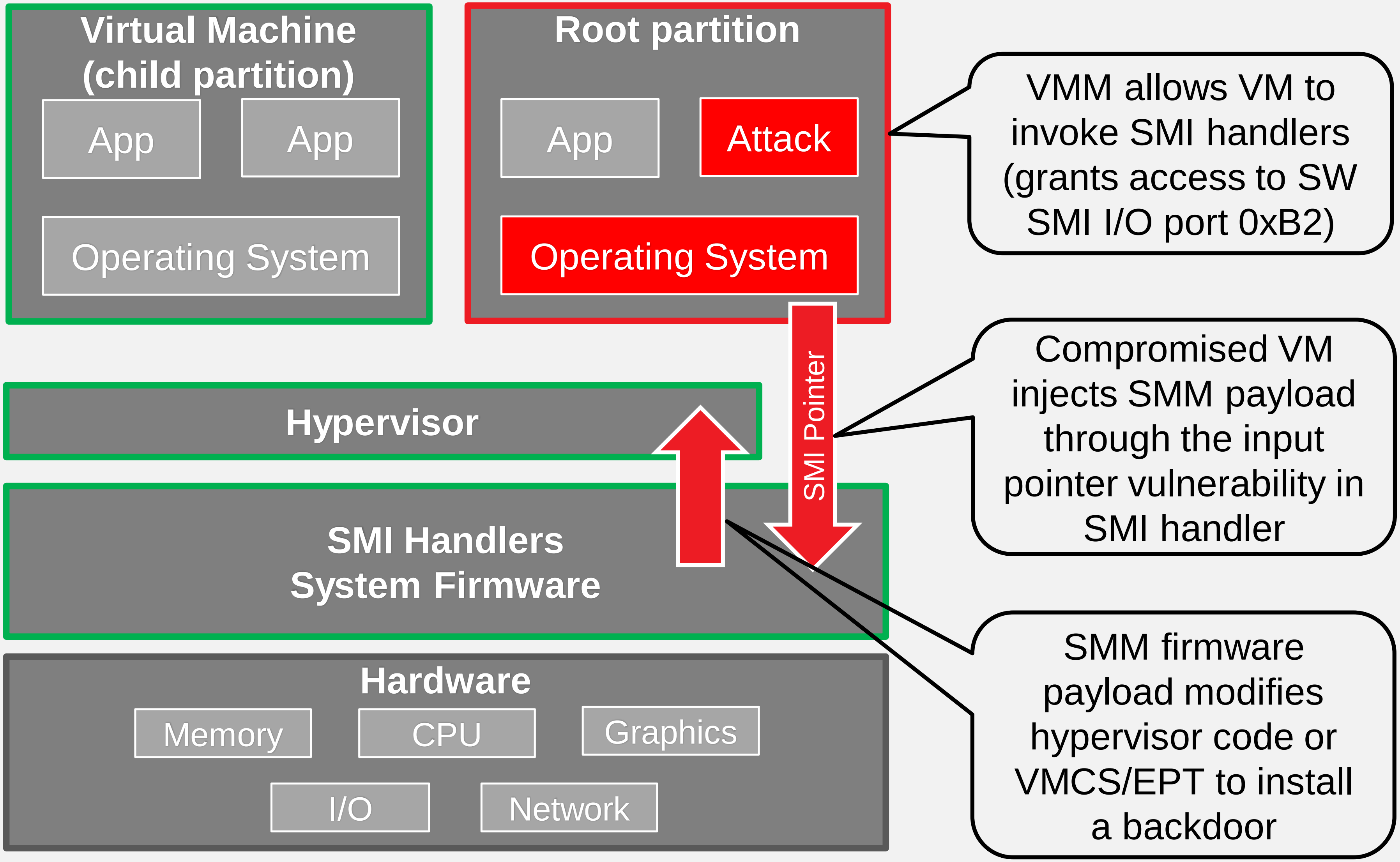
Pointer Vulnerabilities in SMI Handlers

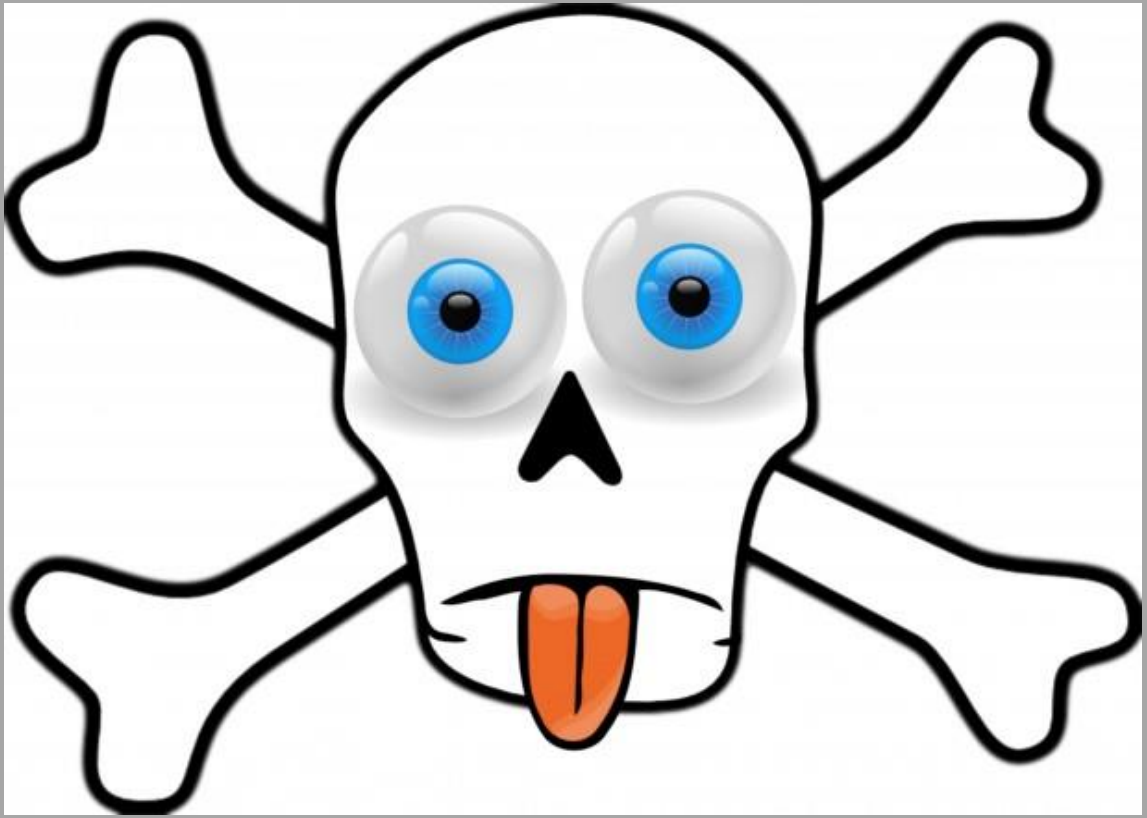


Exploit tricks SMI handler to write to an address **inside SMRAM**

Attacking and Defending BIOS in 2015

Exploiting firmware SMI handler to attack VMM





DEMO

Attacking Hypervisor via
Poisonous Pointers in Firmware SMI handlers

https://youtu.be/zUJEL9cGSE8

Root cause? Port B2h is open to VM in I/O bitmap

```
CPU_BASED_VM_EXEC_CONTROL:
      Bit  2:  0  Interrupt-window exiting
      Bit  3:  1  Use TSC offsetting
      Bit  7:  1  HLT exiting
      Bit  9:  0  INVLPG exiting
      Bit 10:  1  MWAIT exiting
      Bit 11:  1  RDPMC exiting
      Bit 12:  0  RDTSC exiting
      Bit 15:  0  CR3-load exiting
      Bit 16:  0  CR3-store exiting
      Bit 19:  0  CR8-load exiting
      Bit 20:  0  CR8-store exiting
      Bit 21:  1  Use TPR shadow
      Bit 22:  0  NMI-window exiting
      Bit 23:  1  MOV-DR exiting
      Bit 24:  0  Unconditional I/O exiting
      Bit 25:  1  Use I/O bitmaps
      Bit 27:  0  Monitor trap flag
      Bit 28:  1  Use MSR bitmaps
      Bit 29:  1  MONITOR exiting
      Bit 30:  0  PAUSE exiting
      Bit 31:  1  Activate secondary controls

SECONDARY_VM_EXEC_CONTROL:
      Bit  0:  1  Virtualize APIC accesses
      Bit  1:  1  Enable EPT
      Bit  2:  1  Descriptor-table exiting
      Bit  3:  1  Enable RDTSCP
      Bit  4:  0  Virtualize x2APIC mode
```

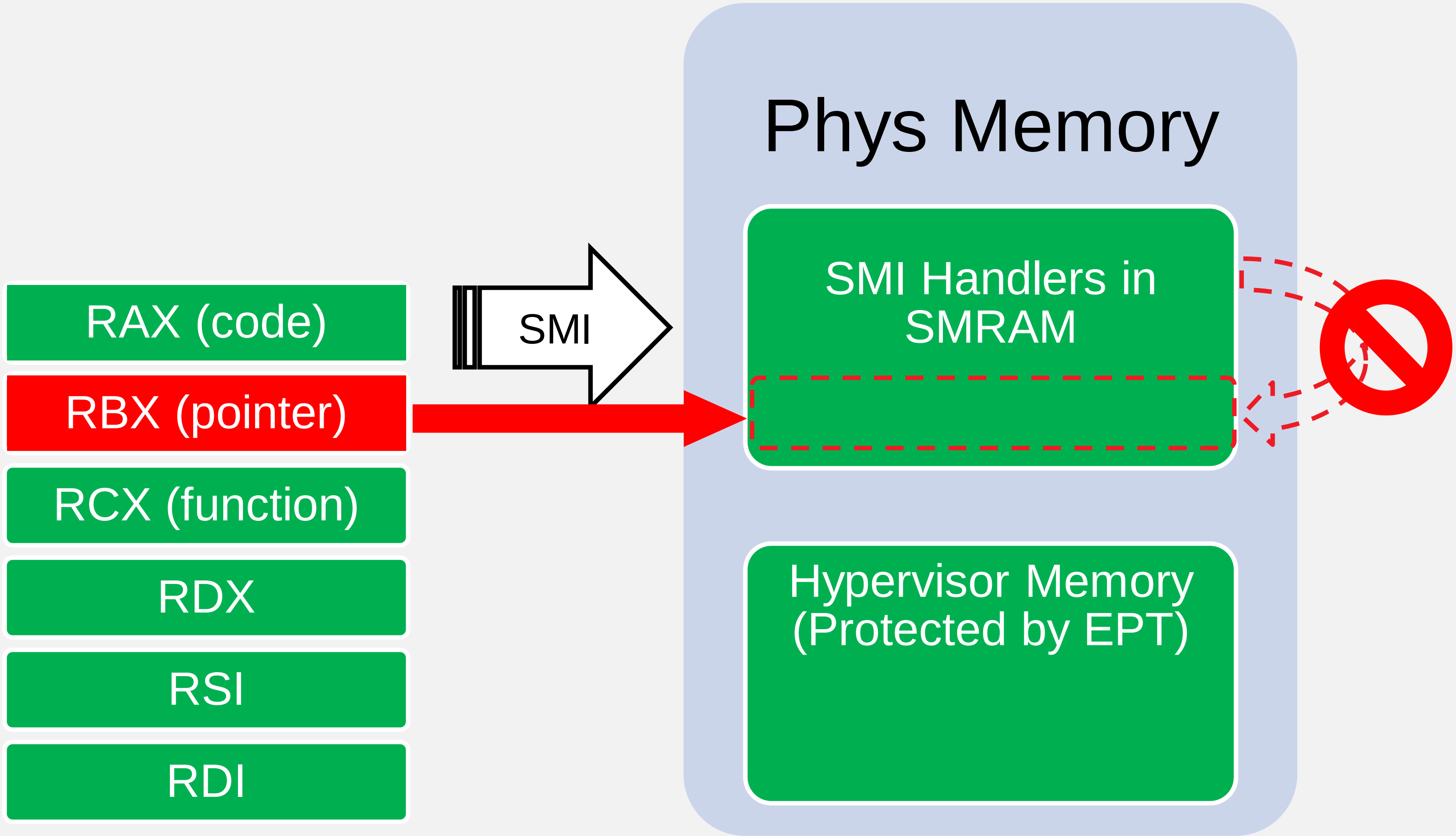
```
IO Bitmap (causes a VM exit):
  0x0020
  0x0021
  0x0064
  0x00a0
  0x00a1
  0x0cf8
  0x0cfc
  0x0cfd
  0x0cfe
  0x0cff

RD MSR Bitmap (doesn't cause a VM exit):
  0x00000174
  0x00000175
  0x00000176
  0xc0000100
  0xc0000101
  0xc0000102

WR MSR Bitmap (doesn't cause a VM exit):
  0x00000174
  0x00000175
  0x00000176
  0xc0000100
  0xc0000101
  0xc0000102
```

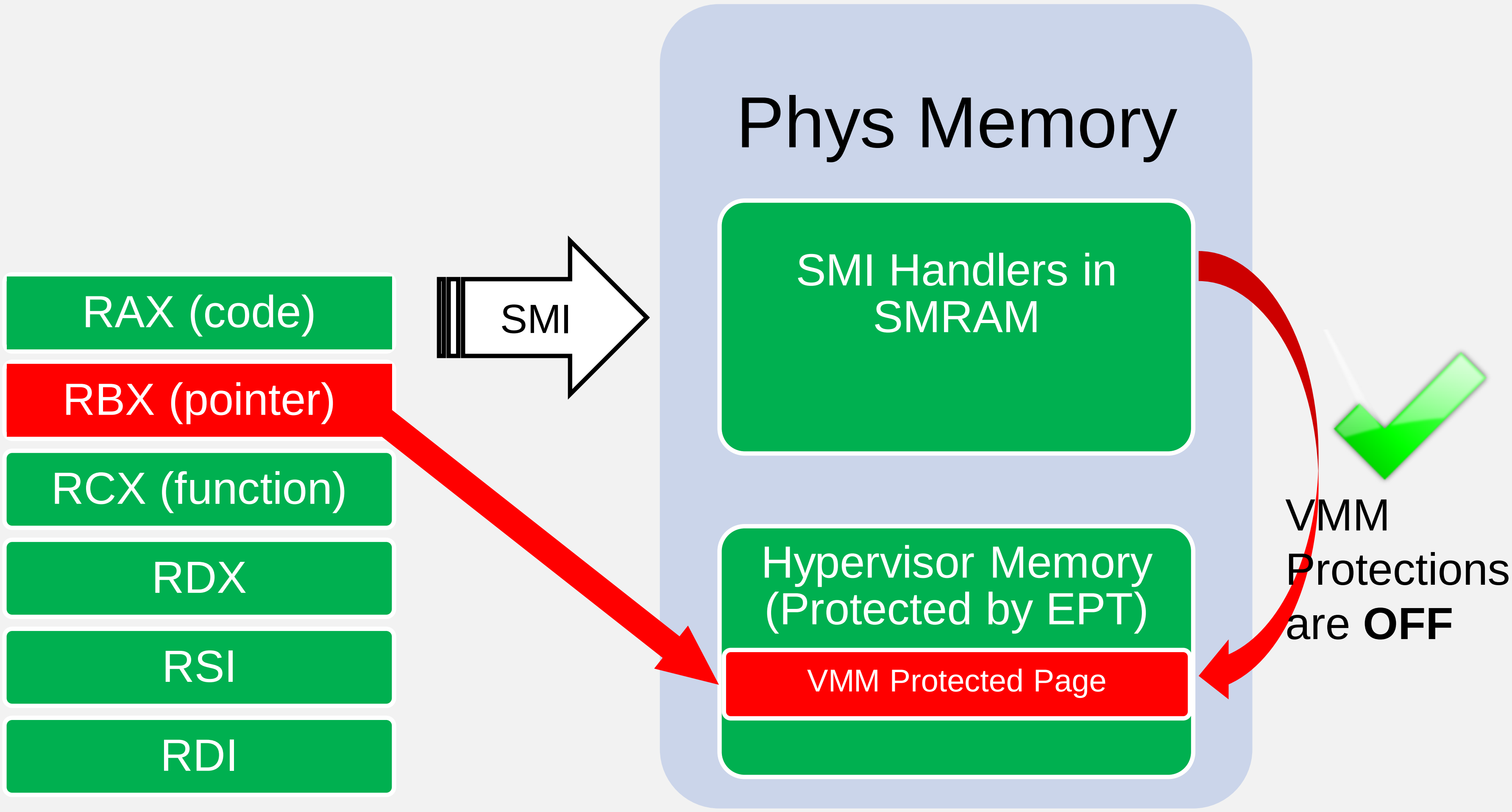
So this is firmware issue, right? What if firmware validates pointers?

Still exploitable…



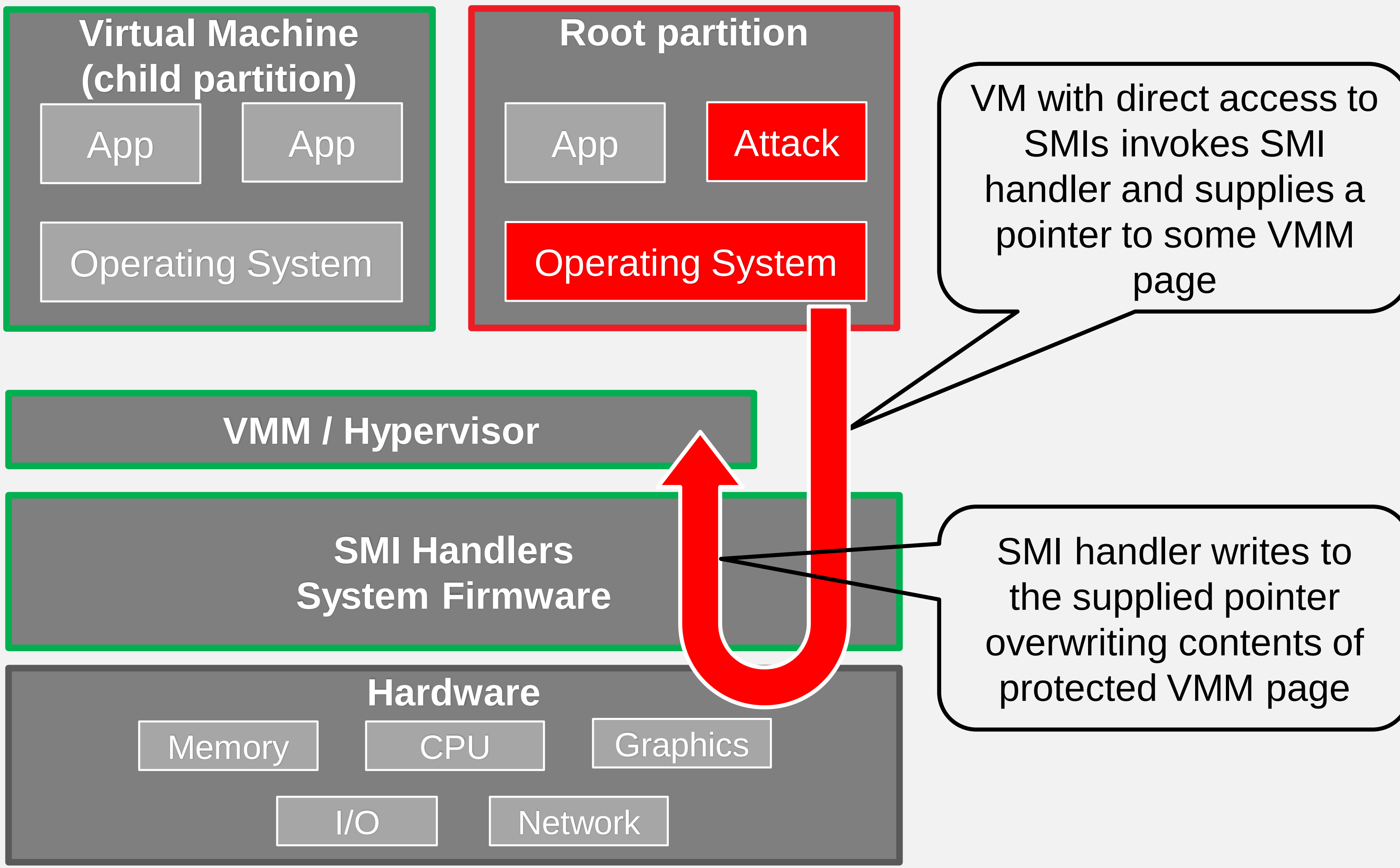
Firmware SMI handler **validates input pointers** to ensure they are outside of SMRAM **preventing overwrite of SMI** code/data

Point SMI handler to overwrite VMM page!



- VT state and EPT protections are OFF in SMM (without STM)
- SMI handler writes to a protected page via supplied pointer

Attacking VMM by proxying through SMI handler

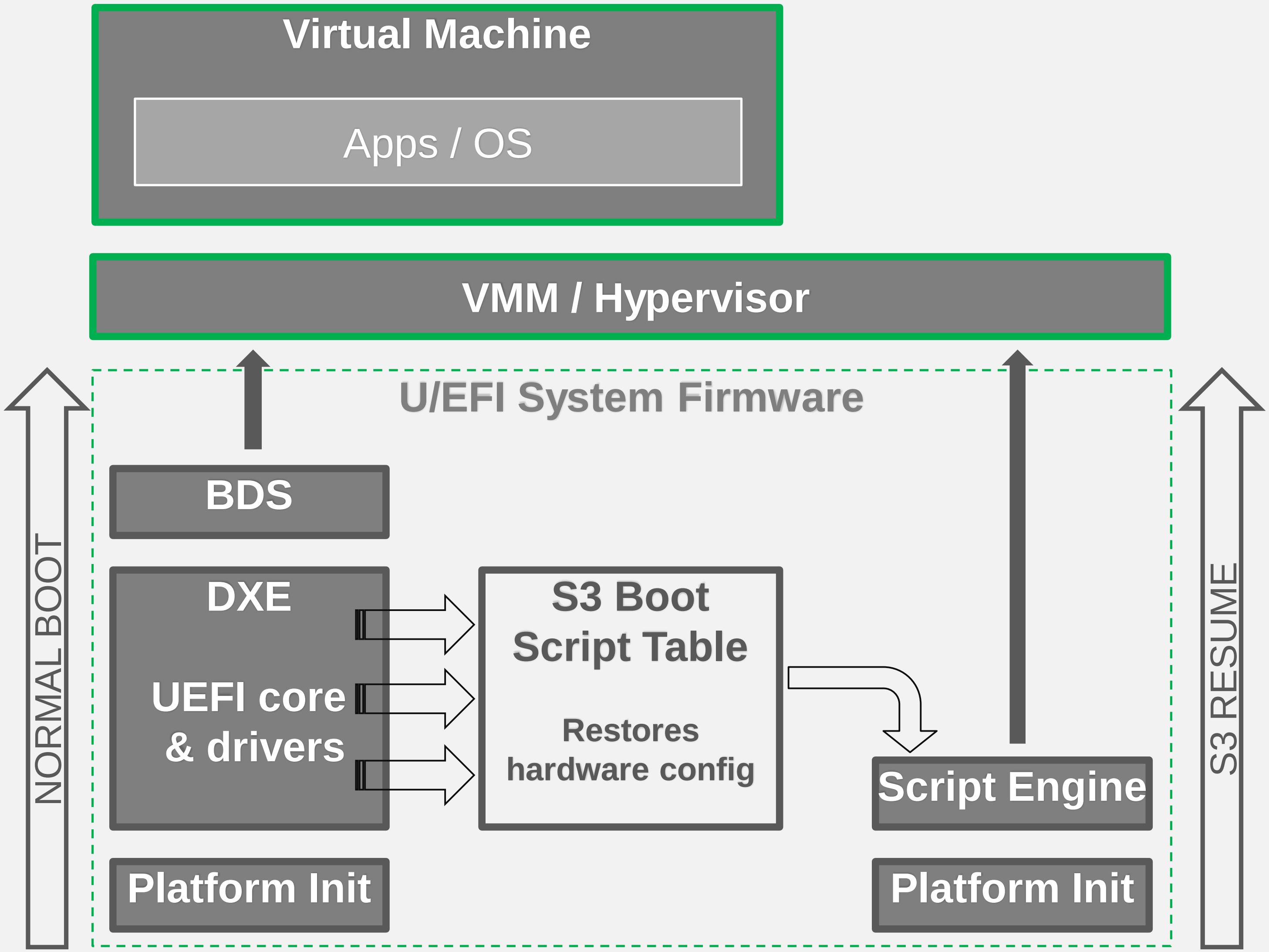


Do Hypervisors Dream of Electric Sheep?

Vulnerability used in this section is VU#976132 a.k.a. S3 Resume Boot Script Vulnerability independently discovered by ATR of Intel Security, Rafal Wojtczuk of Bromium and LegbaCore

It's also used in *Thunderstrike 2* by LegbaCore & Trammell Hudson

Waking the system from S3 “sleep” state



What is S3 boot script table?

A table of opcodes in physical memory which restores platform configuration

`S3_BOOTSCRIPT_MEM_WRITE` opcode writes some value to specified memory location on behalf of firmware

```
[378] Entry at offset 0x31B0 (len = 0x24, header len = 0x8):
Data:
02 02 00 00 00 00 00 00 04 a8 00 e0 00 00 00 00 | ☻☻      ♦J p
01 00 00 00 00 00 00 00 00 38 0e 00             | ☺         8♫
Decoded:
  Opcode : S3_BOOTSCRIPT_MEM_WRITE (0x02)
  Width  : 0x02 (4 bytes)
  Address: 0xE000A804
  Count  : 0x1
  Values : 0x000E3800
```

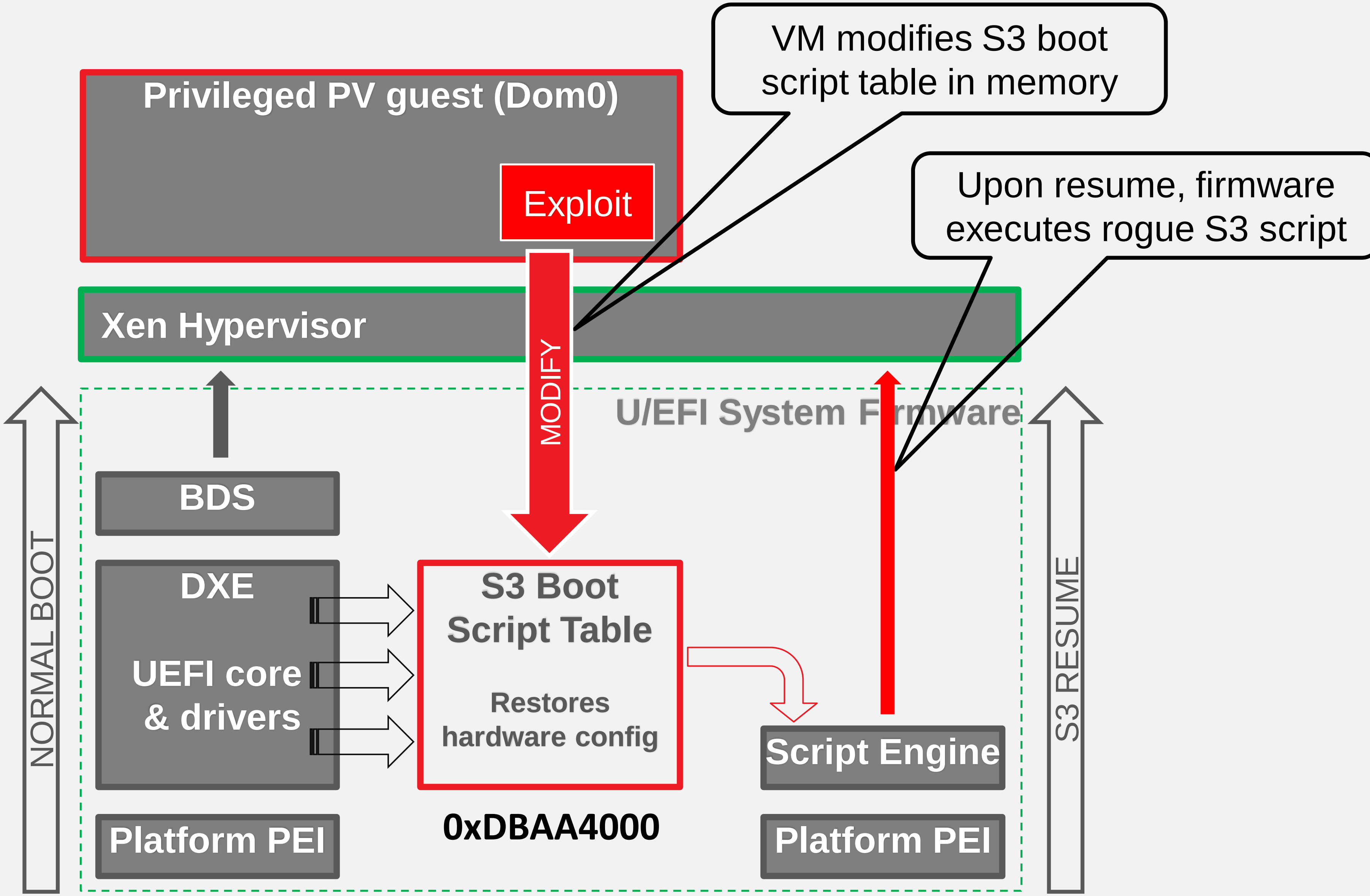
`S3_BOOTSCRIPT_DISPATCH/2`

`S3_BOOTSCRIPT_PCI_CONFIG_WRITE`

`S3_BOOTSCRIPT_IO_WRITE`

…

Xen exposes S3 boot script table to Dom0



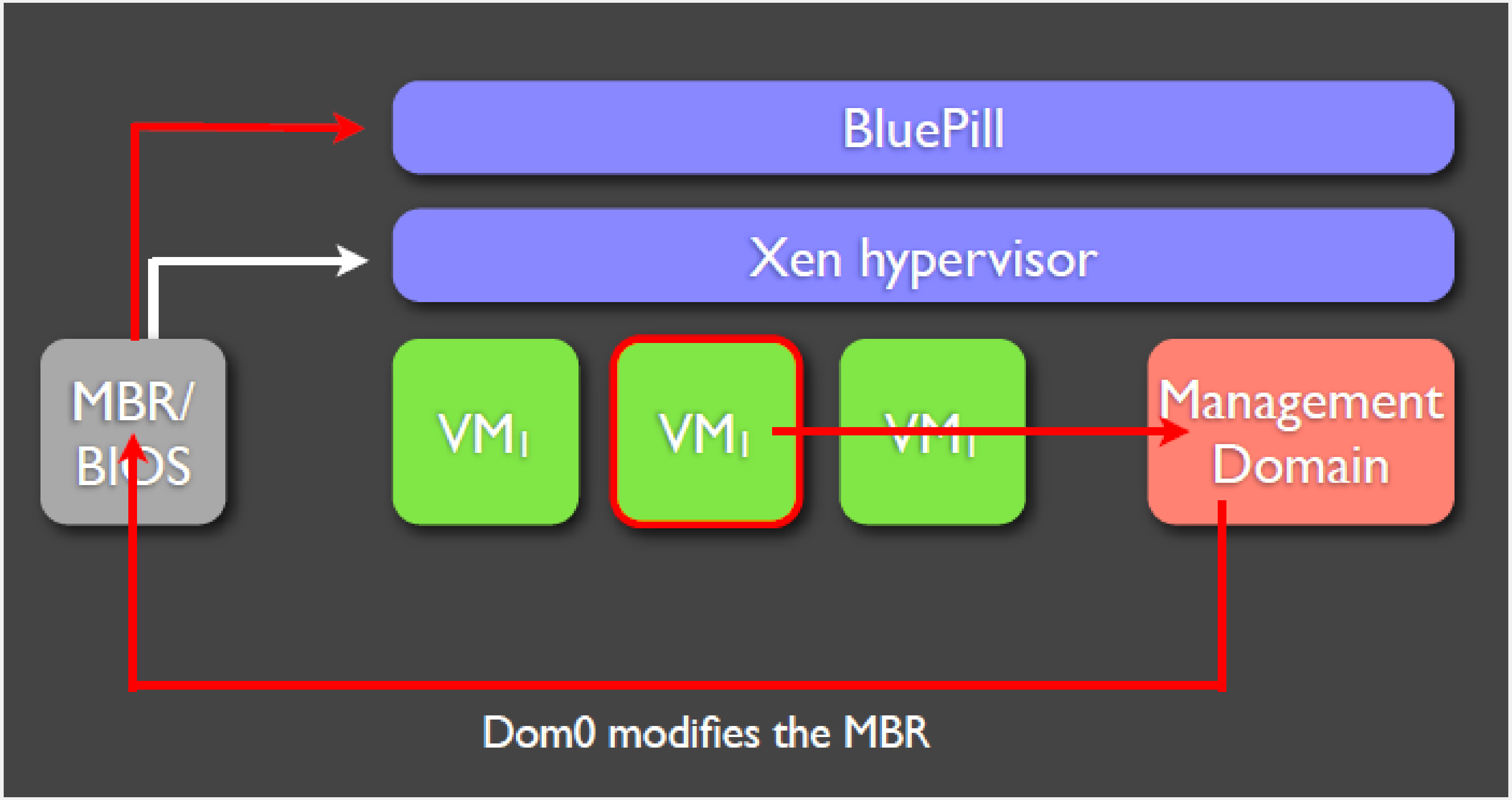
DEMO

Attacking Xen
in its sleep

https://youtu.be/Dsu-scEJyJg

Image source

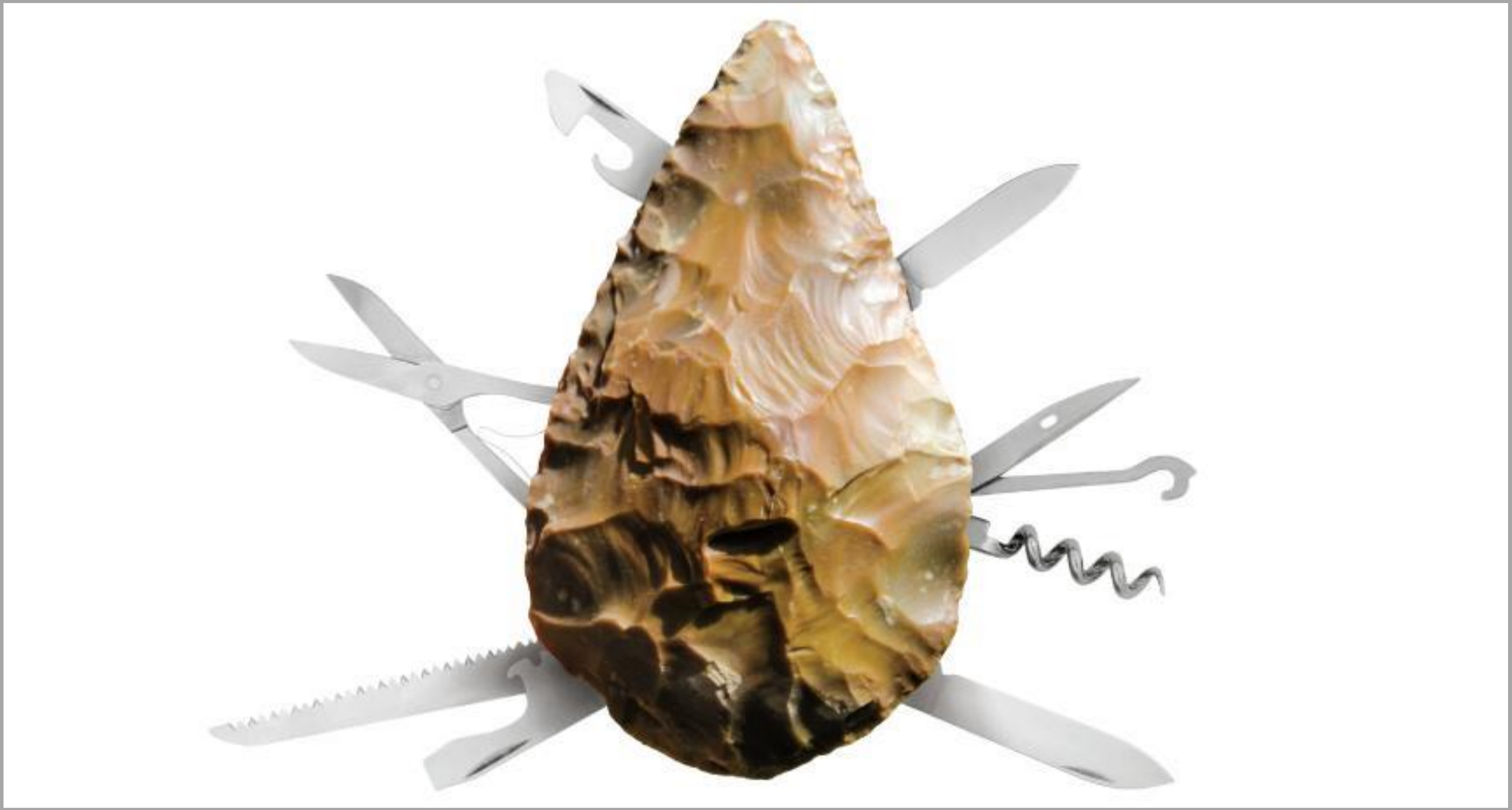
Déjà vu?



Xen 0wning Trilogy (Part 3) by Invisible Things Lab

So these firmware vulnerabilities are exploitable **from privileged guest** (e.g. root partition, Dom0 ..)

What about use cases where guests must be strongly isolated from the root partition?



Tools and Mitigations

Image sciencenews.org

First things first - fix that firmware!

- Firmware **can be tested** for vulnerabilities!

  `common.uefi.s3bootscript`

  (tests S3 boot script protections)

  `tools.smm.smm_ptr`

  (tests for SMI pointer issues)

- Protect the firmware in system flash memory

  `common.bios_wp`

  `common.spi_lock`

  `...`

  (tests firmware protections in system flash memory)

Testing hypervisors…

- Simple hardware emulation fuzzing modules for open source CHIPSEC

  `tools.vmm.*_fuzz`

  I/O, MSR, PCIe device, MMIO overlap, more soon …

- Tools to explore VMM hardware config

  `chipsec_util iommu` (IOMMU)

  `chipsec_util vm` (CPU VM extensions)

Dealing with system firmware attacks..

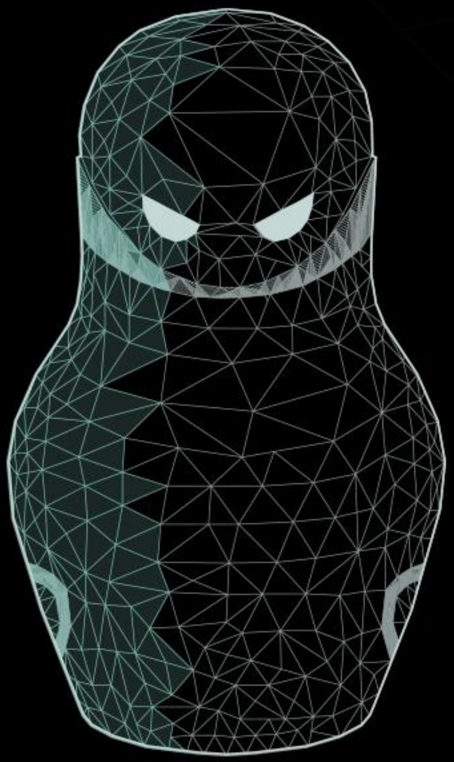
- A number of interfaces through which firmware can be attacked or relay attack onto VMM
  - UEFI variables, SMI handlers, S3 boot script, SPI flash MMIO, FW update..
  - FW doesn’t know memory VMM needs to protect
- VMM need to be careful with which of these it exposes to VMs including to administrative (privileged) guests
  - Some need not be exposed (e.g. S3 boot script), some may be emulated and monitored

Conclusions

- Compromised firmware is bad news for VMM. Test your system’s firmware for security issues
- Windows 10 enables path for firmware deployment via Windows Update
- Secure privileged/administrative guests; attacks from such guests are important
- Vulnerabilities in device and CPU emulation are very common. Fuzz all HW interfaces
- Firmware interfaces/features may affect hypervisor security if exposed to VMs. Both need to be designed to be aware of each other

References

1. CHIPSEC: https://github.com/chipsec/chipsec
2. Intel's ATR Security of System Firmware
3. Attacking and Defending BIOS in 2015 by Intel ATR
4. Hardware Involved Software Attacks by Jeff Forristal
5. Xen 0wning Trilogy by Invisible Things Lab
6. http://www.legbacore.com/Research.html
7. Low level PC attack papers by Xeno Kovah



2015
ZERO NIGHTS

Thank you!