SOURCE

Security Conference And Training

Seattle 2015

Reaching the far corners of MATRIX: generic VMM fingerprinting

Intel Security, Advanced Threat Research (www.intelsecurity.com/atr)

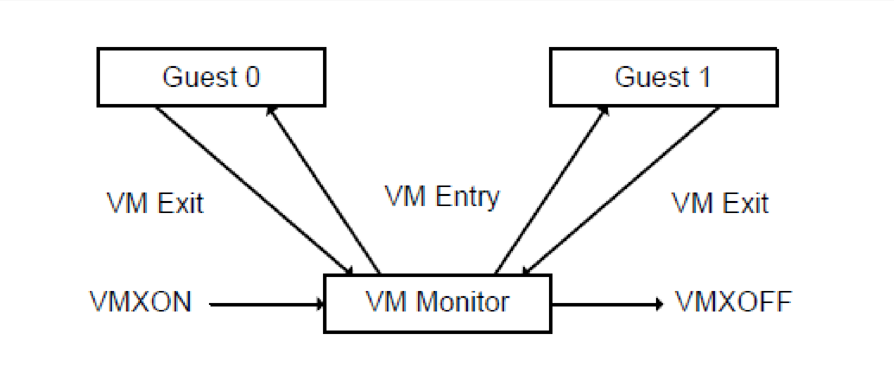
Oleksandr Bazhaniuk, Yuriy Bulygin, Andrew Furtak, Mikhail Gorobets, John Loucaides, Mickey Shkatov

Agenda

- Hypervisor overview
- Current detection techniques
- Hypervisor artifacts detection techniques
- Instruction behavior detection techniques
- CPUID corner cases
- Conclusions

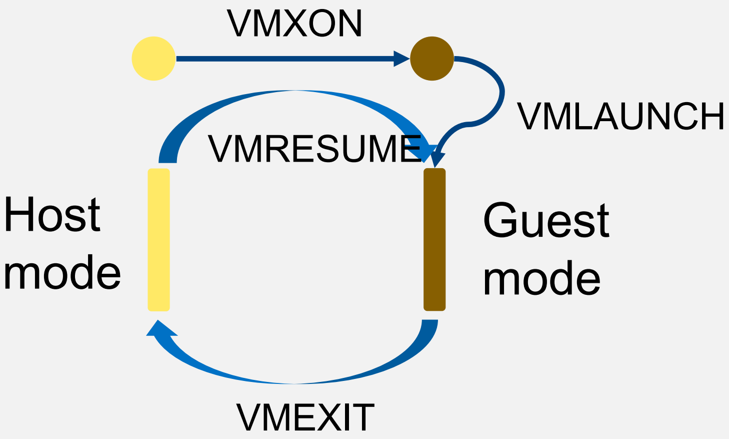
Hypervisor overview

Hypervisor architecture



Hypervisor Code flow:

```
VMXon
Init VMCS
vmlaunch
While(1){
     switch(Exit_code)
     { //VM exit handler intercept
     // within VMM context}
     vmresume
}
VMXoff
```



Hypervisor Isolations

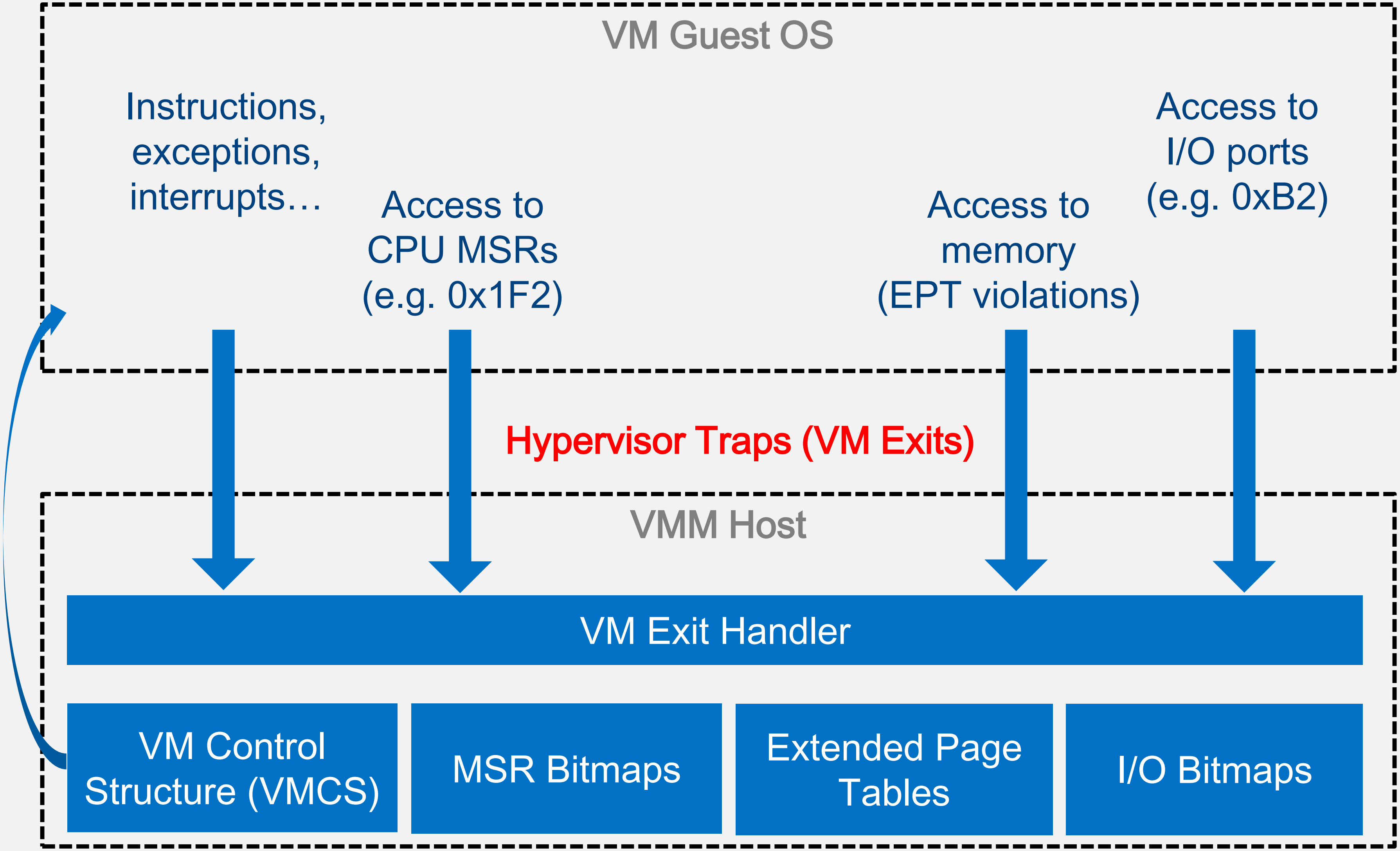
Software Isolation

- **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)



VM Guest OS

Instructions, exceptions, interrupts…

Access to CPU MSRs (e.g. 0x1F2)

Access to memory (EPT violations)

Access to I/O ports (e.g. 0xB2)

Hypervisor Traps (VM Exits)

VMM Host

VM Exit Handler

VM Control Structure (VMCS)

MSR Bitmaps

Extended Page Tables

I/O Bitmaps

Current detection techniques

Official detection techniques. Hyper-V

Hypervisor Top-Level Functional Specification (current 4.0a)

> “On x64 platforms that conform to this specification, this is done by executing the CPUID instruction with an input (EAX) value of 1. Upon execution, code should check bit 31 of register ECX (the “hypervisor present bit”). If this bit is set, a hypervisor is present. In a non-virtualized environment, the bit will be clear.”

- CPUID leaves 0x40000000 - 0x40000006

Leaf	Information Provided	
0x40000000	Hypervisor CPUID leaf range and vendor ID signature.	
	EAX	The maximum input value for hypervisor CPUID information. On Microsoft hypervisors, this will be at least 0x40000005. The vendor ID signature should be used only for reporting and diagnostic purposes.
	EBX	0x7263694D—“Micr”
	ECX	0x666F736F—“osof”
	EDX	0x76482074—“t Hv”

Official detection techniques. VMWare

VMWare KB 1009458

- Bit 31 in ecx of CPUID 0x01 leaf
- CPUID leaves 0x40000000 - 0x400000FF reserved for software use

```
cpuid(0x1, &eax, &ebx, &ecx, &edx);
if (bit 31 of ecx is set) {
     cpuid(0x40000000, &eax, &ebx, &ecx, &edx);
     memcpy(hyper_vendor_id + 0, &ebx, 4);
     memcpy(hyper_vendor_id + 4, &ecx, 4);
     memcpy(hyper_vendor_id + 8, &edx, 4);
     hyper_vendor_id[12] = '\0';
     if (!strcmp(hyper_vendor_id, "VMwareVMware"))
          return 1; // Success - running under VMware
}
return 0;
```

- BIOS serial number starts with “VMware-” or “VMW”
- Hypervisor port

eax = 0x564D5868 (VMware hypervisor magic value)
ebx = 0xFFFFFFFF (UINT_MAX)
ecx = 10 (Getversion command identifier)
edx = 0x5658 (hypervisor port number)

On VMware, this operation modifies the value of register ebx to 0x564D5868

(the VMware hypervisor magic value).

Official detection techniques. KVM and Xen

KVM (arch/x86/include/uapi/asm/kvm_para.h):

```
/* This CPUID returns the signature 'KVMKVMKVM' in ebx, ecx, and edx. It
* should be used to determine that a VM is running under KVM.
*/
#define KVM_CPUID_SIGNATURE 0x40000000
```

Xen (arch-x86/cpuid.h):

```
/*
 * Leaf 1 (0x40000x00)
 * EAX: Largest Xen-information leaf. All leaves up to an including @EAX
 *    are supported by the Xen host.
 * EBX-EDX: "XenVMMXenVMM" signature, allowing positive identification
 *    of a Xen host.
 */
#define XEN_CPUID_SIGNATURE_EBX 0x566e6558 /* "XenV" */
#define XEN_CPUID_SIGNATURE_ECX 0x65584d4d /* "MMXe" */
#define XEN_CPUID_SIGNATURE_EDX 0x4d4d566e /* "nVMM" */
```

Other detection techniques/tools

Timing/benchmarking

- Use instructions that cause VMExit, measure timing difference introduced by hypervisor
- TLB side-channel, Branch prediction side-channel by Edgar Barbosa
- RSB side-channel (hyper-channel) by Yuriy Bulygin

Functional changes, instructions

- Observe different instruction behavior introduced by hypervisor
  - Red Pill
  - ScoopyNG
  - VMDetect
  - virt-what

Implementation peculiarities, artifacts

- Observe typical devices, software, system firmware that is used by hypervisor
  - Scoopy Doo
  - Hyper-V detection script by John Kelbley http://blogs.technet.com/b/tonyso/archive/2009/08/20/hyper-v-how-to-detect-if-you-are-inside-a-vm.aspx
  - Metasploit/checkvm module

Hypervisor artifacts detection techniques

Detection techniques based on artifacts

Implementation peculiarities, artifacts:

- Guest devices
- Drivers/services
- Guest system firmware
- ACPI tables
- Platform information (Platform Manufacturer, Model,…)

Tools to collect artifacts

- Windows internal: Systeminfo, msinfo32, driverquery
- SysInternals: PsInfo, PsService
- Acpidump (from acpica.org)
- DevCon - Device Manager
- CHIPSEC (read MMIO, read physical memory, read ACPI,…)
- RW-everything

Xen artifacts

System information	
System Manufacturer	Xen
System Model	HVM domU
BIOS Version	Xen 4.4.2

Devices	
[CD-ROM] Name	QEMU QEMU DVD-ROM ATA Device
PNP Device ID	IDE\CDROMQEMU_QEMU_DVD-ROM_______________________2.0.____\5&3869DF3D&0&1.0.0
[Disks] Model	QEMU HARDDISK ATA Device

Fields in every ACPI table	
OEM ID	Xen
OEM Table ID	HVM

KVM artifacts

System information	
System Manufacturer	QEMU
Processor	QEMU Virtual CPU version 2.0.0
BIOS Version/Date	Bochs Bochs

Devices	
[CD-ROM] Name	QEMU QEMU DVD-ROM ATA Device
PNP Device ID	IDE\CDROMQEMU_QEMU_DVD-ROM_______________________2.0.____\5&3A2A5854&0&1.0.0
[Disks] Model	QEMU HARDDISK ATA Device

ACPI table	
OEM ID (in every table)	BOCHS
OEM Table ID (in every table)	BXPC+{table_name}

DefinitionBlock ("SSDT.AML", "SSDT”,…)
Scope(_SB.PCI0.ISA)
Device(PEVT)
	Name(_HID, "QEMU0001")

VirtualBox artifacts

System information	
System Manufacturer	innotek GmbH
System Model	VirtualBox
BIOS Version/Date	innotek GmbH VirtualBox

Devices	
[CD-ROM] Name	VBOX CD-ROM ATA Device
PNP Device ID	IDE\CDROMVBOX_CD-ROM_____________________________1.0_____\5&2117B2E5&0&1.0.
[Display] Name	VirtualBox Graphics Adapter
[Disks] Model	VBOX HARDDISK ATA Device

Fields in every ACPI table	
OEM ID	VBOX
OEM Table ID	VBOX+{table_name}

VmWare Player artifacts

System information	
System Manufacturer	VMware, Inc.
System Model	VMware Virtual Platform

Devices	
[CD-ROM] Name	NECVMWar VMware SATA CD01 ATA Device
PNP Device ID	IDE\CDROMNECVMWAR_VMWARE_SATA_CD01_______________1.00____\6&373888B8&0&1.0.0
[Display] Name	VMware SVGA 3D (VMware Virtual SVGA 3D Graphics Adapter, VMware, Inc. compatible)
[Disks] Model	VMware Virtual S SCSI Disk Device
VMware VMCI Bus Device	PCI\VEN_15AD&DEV_0740&SUBSYS_074015AD&REV_10\3&2B8E0B4B&0&3F

Fields in every ACPI table	
OEM ID	VMWARE
OEM Table ID	VMW +{table_name}

Hyper-V artifacts

System information	
System Model	Virtual Machine

Devices	
[CD-ROM] Name	Msft Virtual CD/ROM ATA Device
[Display] Name	Microsoft Virtual Machine Bus Video Device
[Disks] Model	Virtual HD ATA Device
PNP Device ID	VMBUS\{GUID}\5&296C0F0E&0&{GUID}

Fields in every ACPI table	
OEM ID	VRTUAL
OEM Table ID	MICROSFT

Instruction behavior detection techniques

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

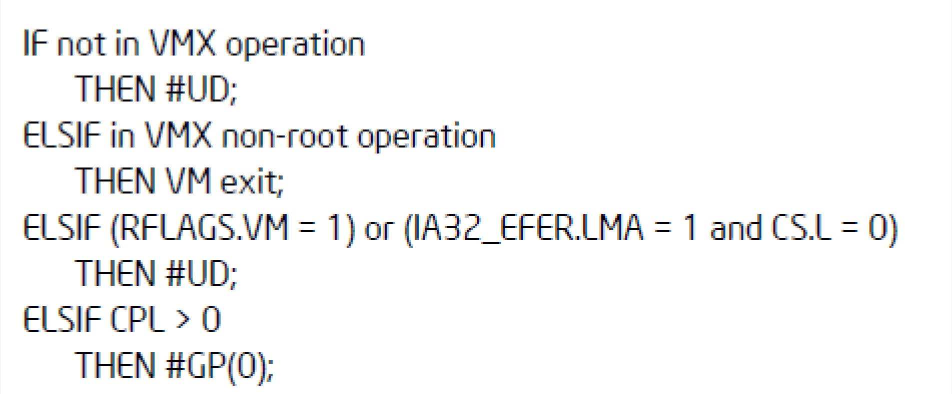
VMExit. Continue

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

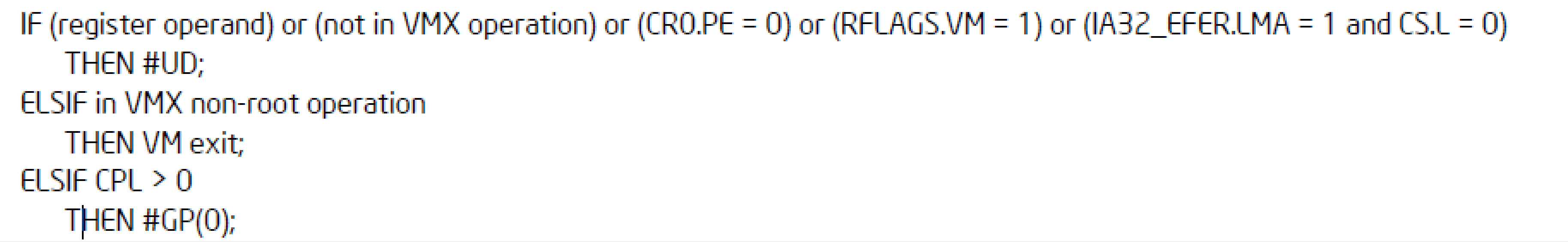
Intel VMX instructions

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

```
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
```

Intel VMX instructions. Host crashing from Ring3

Xen Security Advisory CVE-2013-4551 / XSA-75

Host crash due to guest VMX instruction execution

Permission checks on the emulation paths (intended for guests using nested virtualization) for **VMLAUNCH** and **VMRESUME** were deferred too much. The hypervisor would try to use internal state which is not set up unless nested virtualization is actually enabled for a guest.

This issue was discovered by Jeff Zimmerman.

Intel VMX instructions. Guest crashing from Ring3

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
```

Exceptions and emulated instructions

Exceptions also cause a VMExit.

Hypervisors often use instructions that cause **#UD fault** to implement guest-to-host interface or emulate instructions that are not available on the platform.

Virtual PC: 0F 3F 07 0B

Xen: 0F 0B x e n XXX

where the only supported XXX is CPUID opcode

Hyper-V: 0F 01 C1 C3

on hypercall page only

Exceptions and emulated instructions. AMD SVM

AMD SVM instructions cause #UD fault on Intel platforms

Bare metal Windows 10 x64

```
clgi      : #UD fault
invlpga   : #UD fault
skinit    : #UD fault
stgi      : #UD fault
vmload    : #UD fault
vmmcall   : #UD fault
vmrun     : #UD fault
vmsave    : #UD fault
```

KVM-QEMU x64*,
Windows 10 x64 guest

```
clgi      : #UD fault
invlpga   : #UD fault
skinit    : #UD fault
stgi      : #UD fault
vmload    : #UD fault
vmmcall   : #DE fault
vmrun     : #UD fault
vmsave    : #UD fault
```

Xen 4.4.2 x64,
Windows 10 x64 guest

```
clgi      : #GP fault
invlpga   : #GP fault
skinit    : #GP fault
stgi      : #GP fault
vmload    : #GP fault
vmmcall   : #GP fault
vmrun     : #GP fault
vmsave    : #GP fault
```

* KVM version: QEMU emulator version 2.0.0 (Debian 2.0.0+dfsg-2ubuntu1.19)

Exceptions and emulated instructions. Lock prefix

Lock prefix can be used only with the following instructions:

ADD, ADC, AND, BTC, BTR, BTS, CMPXCHG, CMPXCH8B, CMPXCHG16B, DEC, INC, NEG, NOT, OR, SBB, SUB, XOR, XADD, and XCHG

All other instructions should cause #UD fault if used with lock prefix

Bare metal Windows x64

```
sgdt          : NO EXCEPTION
sgdt_lock     : #UD fault
sidt          : NO EXCEPTION
sidt_lock     : #UD fault
sldt          : NO EXCEPTION
sldt_lock     : #UD fault
str           : NO EXCEPTION
str_lock      : #UD fault
```

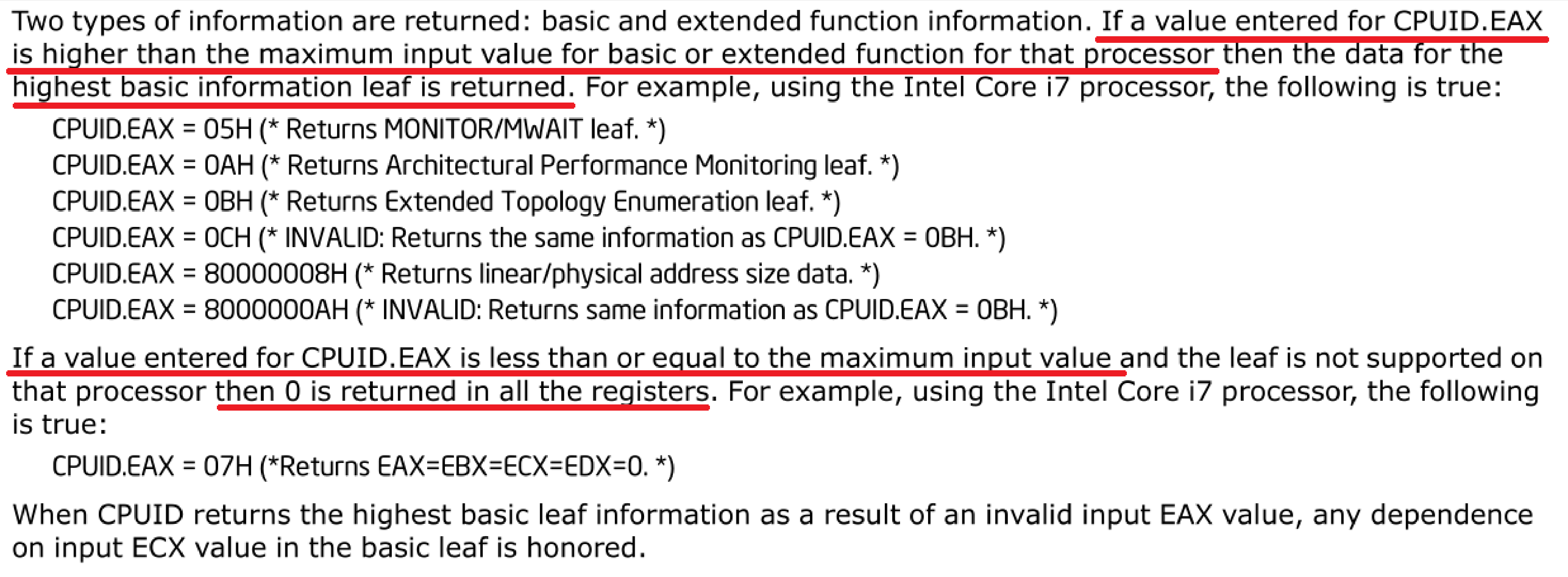
Hyper-V 6.3 (win8.1)
Windows x64 guest

```
sgdt          : NO EXCEPTION
sgdt_lock     : NO EXCEPTION
sidt          : NO EXCEPTION
sidt_lock     : NO EXCEPTION
sldt          : NO EXCEPTION
sldt_lock     : NO EXCEPTION
str           : NO EXCEPTION
str_lock      : NO EXCEPTION
```

CPUID corner cases

CPUID

Hypervisors change CPUID output to reflect guest limitations – less CPU cores, features disabled (debug, tracing, power)



Two types of information are returned: basic and extended function information. If a value entered for CPUID.EAX is higher than the maximum input value for basic or extended function for that processor then the data for the highest basic information leaf is returned. For example, using the Intel Core i7 processor, the following is true:

CPUID.EAX = 05H (* Returns MONITOR/MWAIT leaf. *)
CPUID.EAX = 0AH (* Returns Architectural Performance Monitoring leaf. *)
CPUID.EAX = 0BH (* Returns Extended Topology Enumeration leaf. *)
CPUID.EAX = 0CH (* INVALID: Returns the same information as CPUID.EAX = 0BH. *)
CPUID.EAX = 80000008H (* Returns linear/physical address size data. *)
CPUID.EAX = 8000000AH (* INVALID: Returns same information as CPUID.EAX = 0BH. *)

If a value entered for CPUID.EAX is less than or equal to the maximum input value and the leaf is not supported on that processor then 0 is returned in all the registers. For example, using the Intel Core i7 processor, the following is true:

CPUID.EAX = 07H (*Returns EAX=EBX=ECX=EDX=0. *)

When CPUID returns the highest basic leaf information as a result of an invalid input EAX value, any dependence on input ECX value in the basic leaf is honored.

CPUID. Xen 4.4.2 x64

```
Basic CPUID info
Max input value: 0x0D

cpuid[ 00 ][ 00 ]: eax= 0000000D ebx= 756E6547 ecx= 6C65746E edx= 49656E69
cpuid[ 00 ][ 01 ]: eax= 0000000D ebx= 756E6547 ecx= 6C65746E edx= 49656E69
...
cpuid[ 0D ][ 00 ]: eax= 00000007 ebx= 00000340 ecx= 00000340 edx= 00000000
cpuid[ 0D ][ 01 ]: eax= 00000001 ebx= 00000000 ecx= 00000000 edx= 00000000
cpuid[ 0D ][ 02 ]: eax= 00000100 ebx= 00000240 ecx= 00000000 edx= 00000000
cpuid[ 0D ][ 03 ]: eax= 00000000 ebx= 00000000 ecx= 00000000 edx= 00000000
cpuid[ 0D ][ 04 ]: eax= 00000000 ebx= 00000000 ecx= 00000000 edx= 00000000
cpuid[ 0D ][ 05 ]: eax= 00000000 ebx= 00000000 ecx= 00000000 edx= 00000000

cpuid[ 0E ][ 00 ]: eax= 00000000 ebx= 00000000 ecx= 00000000 edx= 00000000
cpuid[ 0E ][ 01 ]: eax= 00000000 ebx= 00000000 ecx= 00000000 edx= 00000000
cpuid[ 0E ][ 02 ]: eax= 00000000 ebx= 00000000 ecx= 00000000 edx= 00000000
cpuid[ 0E ][ 03 ]: eax= 00000000 ebx= 00000000 ecx= 00000000 edx= 00000000
cpuid[ 0E ][ 04 ]: eax= 00000000 ebx= 00000000 ecx= 00000000 edx= 00000000
cpuid[ 0E ][ 05 ]: eax= 00000000 ebx= 00000000 ecx= 00000000 edx= 00000000
```

Invalid EAX value

CPUID. Hyper-V (latest, win10), the same

Basic CPUID info
Max input value: **0x0D**

```
cpuid[ 00 ][ 00 ]: eax= 0000000D ebx= 756E6547 ecx= 6C65746E edx= 49656E69
cpuid[ 00 ][ 01 ]: eax= 0000000D ebx= 756E6547 ecx= 6C65746E edx= 49656E69
...
cpuid[ 0D ][ 00 ]: eax= 00000007 ebx= 00000340 ecx= 00000340 edx= 00000000
cpuid[ 0D ][ 01 ]: eax= 00000001 ebx= 00000000 ecx= 00000000 edx= 00000000
cpuid[ 0D ][ 02 ]: eax= 00000100 ebx= 00000240 ecx= 00000000 edx= 00000000
cpuid[ 0D ][ 03 ]: eax= 00000000 ebx= 00000000 ecx= 00000000 edx= 00000000
cpuid[ 0D ][ 04 ]: eax= 00000000 ebx= 00000000 ecx= 00000000 edx= 00000000
cpuid[ 0D ][ 05 ]: eax= 00000000 ebx= 00000000 ecx= 00000000 edx= 00000000

cpuid[ 0E ][ 00 ]: eax= 00000000 ebx= 00000000 ecx= 00000000 edx= 00000000
cpuid[ 0E ][ 01 ]: eax= 00000000 ebx= 00000000 ecx= 00000000 edx= 00000000
cpuid[ 0E ][ 02 ]: eax= 00000000 ebx= 00000000 ecx= 00000000 edx= 00000000
cpuid[ 0E ][ 03 ]: eax= 00000000 ebx= 00000000 ecx= 00000000 edx= 00000000
cpuid[ 0E ][ 04 ]: eax= 00000000 ebx= 00000000 ecx= 00000000 edx= 00000000
cpuid[ 0E ][ 05 ]: eax= 00000000 ebx= 00000000 ecx= 00000000 edx= 00000000
```

Invalid EAX value

CPUID. Parallels for Mac

Basic CPUID info
Max input value: **0x0D**

```
cpuid[ 00 ][ 00 ]: eax= 0000000D ebx= 756E6547 ecx= 6C65746E edx= 49656E69
cpuid[ 00 ][ 01 ]: eax= 0000000D ebx= 756E6547 ecx= 6C65746E edx= 49656E69
...
cpuid[ 0D ][ 00 ]: eax= 00000007 ebx= 00000340 ecx= 00000340 edx= 00000000
cpuid[ 0D ][ 01 ]: eax= 00000001 ebx= 00000000 ecx= 00000000 edx= 00000000
cpuid[ 0D ][ 02 ]: eax= 00000100 ebx= 00000240 ecx= 00000000 edx= 00000000
cpuid[ 0D ][ 03 ]: eax= 00000000 ebx= 00000000 ecx= 00000000 edx= 00000000
```

Extended CPUID info
Max input value: **0x80000008**

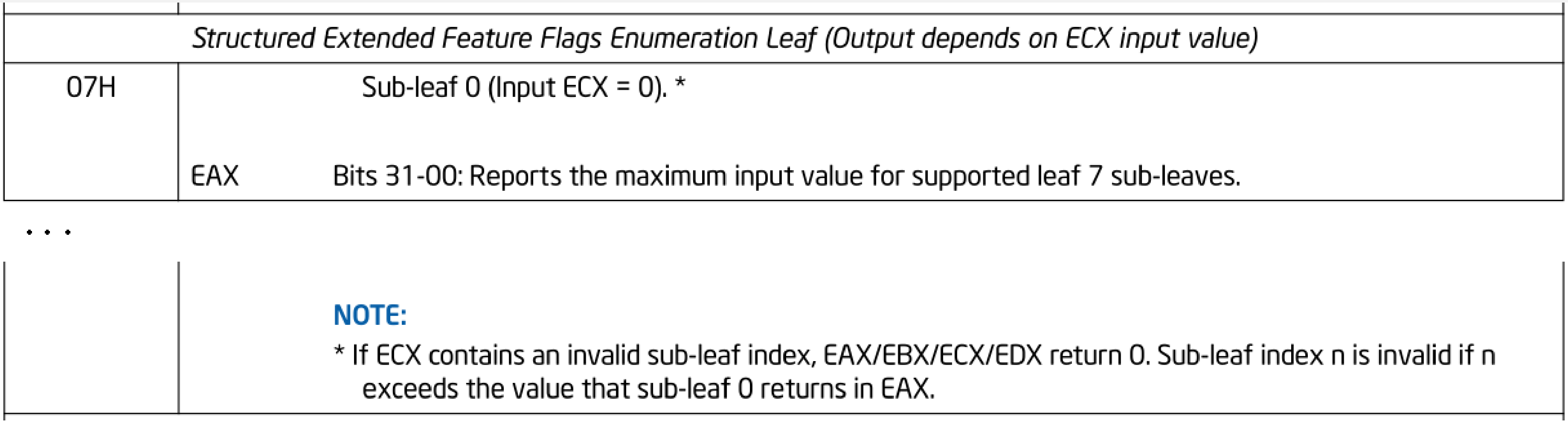
```
...
cpuid[8000000A][ 00 ]: eax= 00000000 ebx= 00000000 ecx= 00000000 edx= 00000000
cpuid[8000000A][ 01 ]: eax= 00000000 ebx= 00000000 ecx= 00000000 edx= 00000000
cpuid[8000000A][ 02 ]: eax= 00000000 ebx= 00000000 ecx= 00000000 edx= 00000000
cpuid[8000000A][ 03 ]: eax= 00000000 ebx= 00000000 ecx= 00000000 edx= 00000000
cpuid[8000000A][ 04 ]: eax= 00000000 ebx= 00000000 ecx= 00000000 edx= 00000000
cpuid[8000000A][ 05 ]: eax= 00000000 ebx= 00000000 ecx= 00000000 edx= 00000000
```

Invalid EAX value

CPUID. VMWare player 6.0.3

Basic CPUID info
Max input value: **0x0D**

```
cpuid[ 00 ][ 00 ]: eax= 0000000D ebx= 756E6547 ecx= 6C65746E edx= 49656E69
cpuid[ 01 ][ 00 ]: eax= 000306A9 ebx= 00010800 ecx= FEBA2203 edx= 0FABFBFF
...
cpuid[ 07 ][ 00 ]: eax= 00000000 ebx= 00000281 ecx= 00000000 edx= 00000000
cpuid[ 07 ][ 01 ]: eax= 00000000 ebx= 00000281 ecx= 00000000 edx= 00000000
cpuid[ 07 ][ 02 ]: eax= 00000000 ebx= 00000281 ecx= 00000000 edx= 00000000
cpuid[ 07 ][ 03 ]: eax= 00000000 ebx= 00000281 ecx= 00000000 edx= 00000000
cpuid[ 07 ][ 04 ]: eax= 00000000 ebx= 00000281 ecx= 00000000 edx= 00000000
...
```



	Structured Extended Feature Flags Enumeration Leaf (Output depends on ECX input value)	
07H		Sub-leaf 0 (Input ECX = 0). *
	EAX	Bits 31-00: Reports the maximum input value for supported leaf 7 sub-leaves.

. . .

		NOTE: * If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n is invalid if n exceeds the value that sub-leaf 0 returns in EAX.

Conclusion

Final thoughts

- Hypervisors can be detected through artifacts
  - If you have malware detection honeypot based on a hypervisor then you should remove all artifacts from the hypervisor environment to prevent honeypot detection.

- Hypervisors can be detected through instruction emulation
  - Hypervisor should check instruction caller privileges during VM exit handler flow.
  - Hypervisor should properly emulate x86 instructions which cause VM exit.

References

- Red Pill... or how to detect VMM using (almost) one CPU instruction
- www.trapkit.de/research/vmm/
- Nate Lawson, Peter Ferrie, Thomas Ptacek. Don't Tell Joanna The Virtualized Rootkit Is Dead. Black Hat USA 2007
- Peter Ferrie. Attacks on More Virtual Machine Emulators
- Peter Ferrie. Attacks on Virtual Machines. Symantec Corporation
- Edgar Barbosa, Blue Pill Detection, COSEINC Advanced Malware Labs, SyScan'07
- Tal Garfinkel, Keith Adams, Andrew Warfield, Jason Franklin. Compatibility is Not Transparency: VMM Detection Myths and Realities. HotOS 2007
- Keith Adams, Blue Pill Detection In Two Easy Steps, July 2007
- CPU side-channels vs. virtualization rootkits: the good, the bad, or the ugly. ToorCon Seattle 2008
- Hyper-V How To: Detect if you are inside a VM
- Hypervisor Top Level Functional Specification for Windows Server 2012 / Windows 8.1

Thank You!

Windows error codes to CPU exceptions

STATUS_ACCESS_VIOLATION	#GP fault
STATUS_PRIVILEGED_INSTRUCTION	#GP fault
STATUS_ILLEGAL_INSTRUCTION	#UD fault
STATUS_INTEGER_DIVIDE_BY_ZERO	#DE fault